

```

1 //<html><details><summary>GShell-0.2.2-HtmlArchive</summary>
2 /*<span id="gsh">
3 <link rel="icon" id="gsh-iconurl" href=""/><!-- place holder -->
4 <meta charset="UTF-8">
5 <meta name="viewport" content="width=device-width, initial-scale=1.0">
6 <title>GShell-0.2.2 by SatoxITS</title>
7 <header id="gsh-banner" height="100px" onclick="shiftBG();" style="">
8 <div align="right"><note>GShell version 0.2.2 // 2020-08-26 // SatoxITS</note></div>
9 </header>
10 <h2>GShell // a General purpose Shell built on the top of Golang</h2>
11 <p>
12 <note>
13 It is a shell for myself, by myself, of myself. --SatoxITS(^-^ )
14 </note>
15 </p>
16 <span id="gsh-WinId" onclick="win_jump('0.1');">0</span>
17 <span id="gsh-menu">
18 | <span id="gsh-menu-exit" onclick="html_close();"></span>
19 | <span id="gsh-menu-fork" onclick="html_fork();">Fork</span>
20 | <span id="gsh-menu-fold" onclick="html_fold(this);">Unfold</span>
21 | <span id="gsh-menu-stop" onclick="html_stop(this,true);">Stop</span>
22 |</span>
23 */
24 /*
25 <details id="overview"><summary>Overview</summary><div class="gsh-src">
26 To be written
27 </div>
28 </details>
29 */
30 /*
31 <details id="gsh-gindex">
32 <summary>Source Code Index</summary><div class="gsh-src" onclick="document.getElementById('gsh-gocode').open=true;">
33 Implementation
34 Structures
35 <a href="#import">import</a>
36 <a href="#struct">struct</a>
37 Main functions
38 <a href="#comexpansion">str-expansion</a> // macro processor
39 <a href="#finder">finder</a> // builtin find + du
40 <a href="#grep">grep</a> // builtin grep + wc + cksum + ...
41 <a href="#plugin">plugin</a> // plugin commands
42 <a href="#ex-commands">system</a> // external commands
43 <a href="#builtin">builtin</a> // builtin commands
44 <a href="#network">network</a> // socket handler
45 <a href="#remote-sh">remote-sh</a> // remote shell
46 <a href="#redirect">redirect</a> // StdIn/Out redirection
47 <a href="#history">history</a> // command history
48 <a href="#rusage">rusage</a> // resource usage
49 <a href="#encode">encode</a> // encode / decode
50 <a href="#IME">IME</a> // command line IME
51 <a href="#getline">getline</a> // line editor
52 <a href="#scanf">scanf</a> // string decomposer
53 <a href="#interpreter">interpreter</a> // command interpreter
54 <a href="#main">main</a>
55 </div>
56 </details>
57 */
58 //<details id="gsh-gocode">
59 //<summary>Source Code</summary><div class="gsh-src" onclick="document.getElementById('gsh-gocode').open=false;">
60 // gsh - Go lang based Shell
61 // (c) 2020 ITS more Co., Ltd.
62 // 2020-0807 created by SatoxITS (sato@its-more.jp)
63
64 package main // gsh main
65 // <a name="import">Imported packages</a> // <a href="https://golang.org/pkg/">Packages</a>
66 import (
67     "fmt" // <a href="https://golang.org/pkg/fmt/">fmt</a>
68     "strings" // <a href="https://golang.org/pkg/strings/">strings</a>
69     "strconv" // <a href="https://golang.org/pkg/strconv/">strconv</a>
70     "sort" // <a href="https://golang.org/pkg/sort/">sort</a>
71     "time" // <a href="https://golang.org/pkg/time/">time</a>
72     "bufio" // <a href="https://golang.org/pkg/bufio/">bufio</a>
73     "io/ioutil" // <a href="https://golang.org/pkg/io/ioutil/">ioutil</a>
74     "os" // <a href="https://golang.org/pkg/os/">os</a>
75     "syscall" // <a href="https://golang.org/pkg/syscall/">syscall</a>
76     "plugin" // <a href="https://golang.org/pkg/plugin/">plugin</a>
77     "net" // <a href="https://golang.org/pkg/net/">net</a>
78     "net/http" // <a href="https://golang.org/pkg/net/http/">http</a>
79     "html" // <a href="https://golang.org/pkg/html/">html</a>
80     "path/filepath" // <a href="https://golang.org/pkg/path/filepath/">filepath</a>
81     "go/types" // <a href="https://golang.org/pkg/go/types/">types</a>
82     "go/token" // <a href="https://golang.org/pkg/go/token/">token</a>
83     "encoding/base64" // <a href="https://golang.org/pkg/encoding/base64/">base64</a>
84     "unicode/utf8" // <a href="https://golang.org/pkg/unicode/utf8/">utf8</a>
85     "gshdata" // gshell's logo and source code
86     "hash/crc32" // <a href="https://golang.org/pkg/unicode/hash/crc32/">crc32</a>
87 )
88 const (
89     NAME = "gsh"
90     VERSION = "0.2.2"
91     DATE = "2020-08-26"
92     AUTHOR = "SatoxITS(^-^)"
93 )
94 var (
95     GSH_HOME = ".gsh" // under home directory
96     GSH_PORT = 9999
97     MaxStreamSize = int64(128*1024*1024*1024) // 128GiB is too large?
98     PROMPT = ">"
99     LINESIZE = (8*1024)
100    PATHSEP = ";" // should be ";" in Windows
101    DIRSEP = "/" // canbe \ in Windows
102 )
103
104 // -xX logging control
105 // --A-- all
106 // --I-- info.
107 // --D-- debug
108 // --T-- time and resource usage
109 // --W-- warning
110 // --E-- error
111 // --F-- fatal error
112 // --Xn- network
113
114 // <a name="struct">Structures</a>
115 type GCommandHistory struct {
116     StartAt time.Time // command line execution started at
117     EndAt time.Time // command line execution ended at
118     ResCode int // exit code of (external command)
119     CmdError error // error string
120     OutData *os.File // output of the command
121     FoundFile []string // output - result of ufind
122     Rusagev [2]syscall.Rusage // Resource consumption, CPU time or so
123     CmdId int // maybe with identified with arguments or impact
124     // redirection commands should not be the CmdId

```

```

125 WorkDir    string    // working directory at start
126 WorkDirX  int      // index in ChdirHistory
127 CmdLine   string    // command line
128 }
129 type GChdirHistory struct {
130     Dir      string
131     MovedAt time.Time
132     CmdIndex int
133 }
134 type CmdMode struct {
135     Background bool
136 }
137 type PluginInfo struct {
138     Spec      *plugin.Plugin
139     Addr      plugin.Symbol
140     Name      string // maybe relative
141     Path      string // this is in Plugin but hidden
142 }
143 type GServer struct {
144     host      string
145     port      string
146 }
147
148 // <a href="https://tools.ietf.org/html/rfc3230">Digest</a>
149 const ( // SumType
150     SUM_ITEMS    = 0x000001 // items count
151     SUM_SIZE     = 0x000002 // data length (simply added)
152     SUM_SIZEHASH = 0x000004 // data length (hashed sequence)
153     SUM_DATEHASH = 0x000008 // date of data (hashed sequence)
154     // also envelope attributes like time stamp can be a part of digest
155     // hashed value of sizes or mod-date of files will be useful to detect changes
156
157     SUM_WORDS    = 0x000010 // word count is a kind of digest
158     SUM_LINES   = 0x000020 // line count is a kind of digest
159     SUM_SUM64   = 0x000040 // simple add of bytes, useful for human too
160
161     SUM_SUM32_BITS = 0x000100 // the number of true bits
162     SUM_SUM32_2BYTE = 0x000200 // 16bits words
163     SUM_SUM32_4BYTE = 0x000400 // 32bits words
164     SUM_SUM32_8BYTE = 0x000800 // 64bits words
165
166     SUM_SUM16_BSD = 0x001000 // UNIXsum -sum -bsd
167     SUM_SUM16_SYSV = 0x002000 // UNIXsum -sum -sysv
168     SUM_UNIXFILE  = 0x004000
169     SUM_CRCIEEE  = 0x008000
170 )
171 type CheckSum struct {
172     Files    int64 // the number of files (or data)
173     Size     int64 // content size
174     Words    int64 // word count
175     Lines    int64 // line count
176     SumType  int
177     Sum64    uint64
178     Crc32Table  crc32.Table
179     Crc32Val    uint32
180     Sum16       int
181     Ctime       time.Time
182     Atime       time.Time
183     Mtime       time.Time
184     Start       time.Time
185     Done        time.Time
186     RusgAtStart [2]syscall.Rusage
187     RusgAtEnd   [2]syscall.Rusage
188 }
189 type ValueStack [][]string
190 type GshContext struct {
191     StartDir string // the current directory at the start
192     GetLine  string // gsh-getline command as a input line editor
193     ChdirHistory []GChdirHistory // the 1st entry is wd at the start
194     gshPA      syscall.ProcAttr
195     CommandHistory []GCommandHistory
196     CmdCurrent   GCommandHistory
197     Background   bool
198     BackgroundJobs []int
199     LastRusage   syscall.Rusage
200     GshHomeDir   string
201     TerminalId   int
202     CmdTrace     bool // should be [map]
203     CmdTime      bool // should be [map]
204     PluginFuncs []PluginInfo
205     iValues      []string
206     iDelimiter   string // field separator of print out
207     iFormat      string // default print format (of integer)
208     iValStack    ValueStack
209     LastServer   GServer
210     RSERVER      string // [gsh://]host[:port]
211     RWD          string // remote (target, there) working directory
212     lastCheckSum CheckSum
213 }
214
215 func nsleep(ns time.Duration){
216     time.Sleep(ns)
217 }
218 func usleep(ns time.Duration){
219     nsleep(ns*1000)
220 }
221 func msleep(ns time.Duration){
222     nsleep(ns*1000000)
223 }
224 func sleep(ns time.Duration){
225     nsleep(ns*1000000000)
226 }
227
228 func strBegins(str, pat string)(bool){
229     if len(pat) <= len(str){
230         yes := str[0:len(pat)] == pat
231         //fmt.Printf("--D-- strBegins(%v,%v)=%v\n",str,pat,yes)
232         return yes
233     }
234     //fmt.Printf("--D-- strBegins(%v,%v)=%v\n",str,pat,false)
235     return false
236 }
237 func isin(what string, list []string) bool {
238     for _, v := range list {
239         if v == what {
240             return true
241         }
242     }
243     return false
244 }
245 func isinX(what string,list[]string)(int){
246     for i,v := range list {
247         if v == what {
248             return i
249         }
250     }

```

```

250     }
251     return -1
252 }
253
254 func env(opts []string) {
255     env := os.Environ()
256     if !isIn("-s", opts){
257         sort.Slice(env, func(i,j int) bool {
258             return env[i] < env[j]
259         })
260     }
261     for _, v := range env {
262         fmt.Printf("%v\n",v)
263     }
264 }
265
266 // - rewriting should be context dependent
267 // - should postpone until the real point of evaluation
268 // - should rewrite only known notation of symbol
269 func scanInt(str string)(val int,leng int){
270     leng = -1
271     for i,ch := range str {
272         if '0' <= ch && ch <= '9' {
273             leng = i+1
274         }else{
275             break
276         }
277     }
278     if 0 < leng {
279         ival,_ := strconv.Atoi(str[0:leng])
280         return ival,leng
281     }else{
282         return 0,0
283     }
284 }
285 func substHistory(gshCtx *GshContext,str string,i int,rstr string)(leng int,rst string){
286     if len(str[i+1:]) == 0 {
287         return 0,rstr
288     }
289     hi := 0
290     histlen := len(gshCtx.CommandHistory)
291     if str[i+1] == '!' {
292         hi = histlen - 1
293         leng = 1
294     }else{
295         hi,leng = scanInt(str[i+1:])
296         if leng == 0 {
297             return 0,rstr
298         }
299         if hi < 0 {
300             hi = histlen + hi
301         }
302     }
303     if 0 <= hi && hi < histlen {
304         var ext byte
305         if 1 < len(str[i+leng:]) {
306             ext = str[i+leng:][1]
307         }
308         //fmt.Printf("--D-- %v(%c)\n",str[i+leng:],str[i+leng])
309         if ext == 'f' {
310             leng += 1
311             xlist := []string{}
312             list := gshCtx.CommandHistory[hi].FoundFile
313             for _,v := range list {
314                 //list[i] = escapeWhiteSP(v)
315                 xlist = append(xlist,escapeWhiteSP(v))
316             }
317             //rstr += strings.Join(list," ")
318             rstr += strings.Join(xlist," ")
319         }else{
320             if ext == 'e' || ext == 'd' {
321                 // !N0 .. workdir at the start of the command
322                 leng += 1
323                 rstr += gshCtx.CommandHistory[hi].WorkDir
324             }else{
325                 rstr += gshCtx.CommandHistory[hi].CmdLine
326             }
327         }else{
328             leng = 0
329         }
330     }
331     return leng,rstr
332 }
333 func escapeWhiteSP(str string)(string){
334     if len(str) == 0 {
335         return "\\z" // empty, to be ignored
336     }
337     rstr := ""
338     for _,ch := range str {
339         switch ch {
340             case '\\': rstr += "\\\\"
341             case ' ': rstr += "\\s"
342             case '\t': rstr += "\\t"
343             case '\r': rstr += "\\r"
344             case '\n': rstr += "\\n"
345             default: rstr += string(ch)
346         }
347     }
348     return rstr
349 }
350 func unescapeWhiteSP(str string)(string){ // strip original escapes
351     rstr := ""
352     for i := 0; i < len(str); i++ {
353         ch := str[i]
354         if ch == '\\' {
355             if i+1 < len(str) {
356                 switch str[i+1] {
357                     case 'z':
358                         continue;
359                 }
360             }
361             rstr += string(ch)
362         }
363     }
364     return rstr
365 }
366 func unescapeWhiteSPV(strv []string)([]string){ // strip original escapes
367     ustrv := []string{}
368     for _,v := range strv {
369         ustrv = append(ustrv,unescapeWhiteSP(v))
370     }
371     return ustrv
372 }
373 // <a name="comexpansion">str-expansion</a>
374 // - this should be a macro processor

```

```

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

```

```

500
501 // <a name="encode">Encode / Decode</a>
502 // <a href="https://golang.org/pkg/encoding/base64/#example_NewEncoder">Encoder</a>
503 func (gshCtx *GshContext)Enc(argv[]string){
504     file := os.Stdin
505     buff := make([]byte,LINESIZE)
506     li := 0
507     encoder := base64.NewEncoder(base64.StdEncoding,os.Stdout)
508     for li = 0; ; li++ {
509         count, err := file.Read(buff)
510         if count <= 0 {
511             break
512         }
513         if err != nil {
514             break
515         }
516         encoder.Write(buff[0:count])
517     }
518     encoder.Close()
519 }
520 func (gshCtx *GshContext)Dec(argv[]string){
521     decoder := base64.NewDecoder(base64.StdEncoding,os.Stdin)
522     li := 0
523     buff := make([]byte,LINESIZE)
524     for li = 0; ; li++ {
525         count, err := decoder.Read(buff)
526         if count <= 0 {
527             break
528         }
529         if err != nil {
530             break
531         }
532         os.Stdout.Write(buff[0:count])
533     }
534 }
535 // lnspl [N] [-crlf][-C \\\]
536 func (gshCtx *GshContext)SplitLine(argv[]string){
537     reader := bufio.NewReaderSize(os.Stdin,64*1024)
538     ni := 0
539     toi := 0
540     for ni = 0; ; ni++ {
541         line, err := reader.ReadString('\n')
542         if len(line) <= 0 {
543             if err != nil {
544                 fmt.Fprintf(os.Stderr,"--I-- lnspl %d to %d (%v)\n",ni,toi,err)
545                 break
546             }
547         }
548         off := 0
549         ilen := len(line)
550         remlen := len(line)
551         for oi := 0; oi < remlen; oi++ {
552             olen := remlen
553             addnl := false
554             if 72 < olen {
555                 olen = 72
556                 addnl = true
557             }
558             fmt.Fprintf(os.Stderr,"--D-- write %d [%d.%d] %d %d/%d/%d\n",
559                 toi,ni,oi,off,olen,remlen,ilen)
560             toi += 1
561             os.Stdout.Write([]byte(line[0:olen]))
562             if addnl {
563                 //os.Stdout.Write([]byte("\r\n"))
564                 os.Stdout.Write([]byte("\n"))
565                 os.Stdout.Write([]byte("\n"))
566             }
567             line = line[olen:]
568             off += olen
569             remlen -= olen
570         }
571     }
572     fmt.Fprintf(os.Stderr,"--I-- lnspl %d to %d\n",ni,toi)
573 }
574
575 // CRC32 <a href="http://golang.jp/pkg/hash-crc32">crc32</a>
576 // 1 0000 0100 1100 0001 0001 1101 1011 0111
577 var CRC32UNIX uint32 = uint32(0x04C11DB7) // Unix cksum
578 var CRC32IEEE uint32 = uint32(0xEDB88320)
579 func byteCRC32add(crc uint32,str[]byte,len uint64)(uint32){
580     var i uint64
581     for i = 0; i < len; i++ {
582         var oct = str[i]
583         for bi := 0; bi < 8; bi++ {
584             ovf1 := (crc & 0x80000000) != 0
585             ovf2 := (oct & 0x80) != 0
586             ovf := (ovf1 && !ovf2) || (!ovf1 && ovf2)
587             oct <<= 1
588             crc <<= 1
589             if ovf { crc ^= CRC32UNIX }
590         }
591     }
592     return crc;
593 }
594 func byteCRC32end(crc uint32, len uint64)(uint32){
595     var slen = make([]byte,4)
596     var li = 0
597     for li = 0; li < 4; {
598         slen[li] = byte(len)
599         li += 1
600         len >>= 8
601         if( len == 0 ){
602             break
603         }
604     }
605     crc = byteCRC32add(crc,slen,uint64(li))
606     crc ^= 0xFFFFFFFF
607     return crc
608 }
609 func byteCRC32(str[]byte,len uint64)(crc uint32){
610     crc = byteCRC32add(0,str,len)
611     crc = byteCRC32end(crc,len)
612     return crc
613 }
614 func CRC32Finish(crc uint32, table *crc32.Table, len uint64)(uint32){
615     var slen = make([]byte,4)
616     var li = 0
617     for li = 0; li < 4; {
618         slen[li] = byte(len & 0xFF)
619         li += 1
620         len >>= 8
621         if( len == 0 ){
622             break
623         }
624     }

```

```

625     crc = crc32.Update(crc,table,slen)
626     crc ^= 0xFFFFFFFF
627     return crc
628 }
629
630 func (gsh*GshContext)xChecksum(path string,argv[]string, sum*Checksum)(int64){
631     if isin("-type/f",argv) && !IsRegFile(path){
632         return 0
633     }
634     if isin("-type/d",argv) && IsRegFile(path){
635         return 0
636     }
637     file, err := os.OpenFile(path,os.O_RDONLY,0)
638     if err != nil {
639         fmt.Printf("--E-- cksum %v (%v)\n",path,err)
640         return -1
641     }
642     defer file.Close()
643     if gsh.CmdTrace { fmt.Printf("--I-- cksum %v %v\n",path,argv) }
644
645     bi := 0
646     var buff = make([]byte,32*1024)
647     var total int64 = 0
648     var initTime = time.Time{}
649     if sum.Start == initTime {
650         sum.Start = time.Now()
651     }
652     for bi = 0; ; bi++ {
653         count,err := file.Read(buff)
654         if count <= 0 || err != nil {
655             break
656         }
657         if (sum.SumType & SUM_SUM64) != 0 {
658             s := sum.Sum64
659             for _,c := range buff[0:count] {
660                 s += uint64(c)
661             }
662             sum.Sum64 = s
663         }
664         if (sum.SumType & SUM_UNIXFILE) != 0 {
665             sum.Crc32Val = byteCRC32add(sum.Crc32Val,buff,uint64(count))
666         }
667         if (sum.SumType & SUM_CRCIEEE) != 0 {
668             sum.Crc32Val = crc32.Update(sum.Crc32Val,&sum.Crc32Table,buff[0:count])
669         }
670         // <a href="https://en.wikipedia.org/wiki/BSD_checksum">BSD checksum</a>
671         if (sum.SumType & SUM_SUM16_BSD) != 0 {
672             s := sum.Sum16
673             for _,c := range buff[0:count] {
674                 s = (s >> 1) + ((s & 1) << 15)
675                 s += int(c)
676                 s &= 0xFFFF
677                 //fmt.Printf("BSDsum: %d[%d] %d\n",sum.Size+int64(i),i,s)
678             }
679             sum.Sum16 = s
680         }
681         if (sum.SumType & SUM_SUM16_SYSV) != 0 {
682             for bj := 0; bj < count; bj++ {
683                 sum.Sum16 += int(buff[bj])
684             }
685         }
686         total += int64(count)
687     }
688     sum.Done = time.Now()
689     sum.Files += 1
690     sum.Size += total
691     if !isin("-s",argv) {
692         fmt.Printf("%v ",total)
693     }
694     return 0
695 }
696
697 // <a name="grep">grep</a>
698 // "lines", "lin" or "lnp" for "(text) line processor" or "scanner"
699 // a*,lab,c, ... sequential combination of patterns
700 // what "LINE" is should be definable
701 // generic line-by-line processing
702 // grep [-v]
703 // cat -n -v
704 // uniq [-c]
705 // tail -f
706 // sed s/x/y/ or awk
707 // grep with line count like wc
708 // rewrite contents if specified
709 func (gsh*GshContext)xGrep(path string,rxpv[]string)(int){
710     file, err := os.OpenFile(path,os.O_RDONLY,0)
711     if err != nil {
712         fmt.Printf("--E-- grep %v (%v)\n",path,err)
713         return -1
714     }
715     defer file.Close()
716     if gsh.CmdTrace { fmt.Printf("--I-- grep %v %v\n",path,rxpv) }
717     //reader := bufio.NewReaderSize(file,LINESIZE)
718     reader := bufio.NewReaderSize(file,80)
719     li := 0
720     found := 0
721     for li = 0; ; li++ {
722         line, err := reader.ReadString('\n')
723         if len(line) <= 0 {
724             break
725         }
726         if 150 < len(line) {
727             // maybe binary
728             break;
729         }
730         if err != nil {
731             break
732         }
733         if 0 <= strings.Index(string(line),rxpv[0]) {
734             found += 1
735             fmt.Printf("%s:%d: %s",path,li,line)
736         }
737     }
738     //fmt.Printf("total %d lines %s\n",li,path)
739     //if( 0 < found ){ fmt.Printf("(found %d lines %s)\n",found,path); }
740     return found
741 }
742
743 // <a name="finder">Finder</a>
744 // finding files with it name and contents
745 // file names are ORED
746 // show the content with %x fmt list
747 // ls -R
748 // tar command by adding output
749 type fileSum struct {

```

```

750 Err int64 // access error or so
751 Size int64 // content size
752 DupSize int64 // content size from hard links
753 Blocks int64 // number of blocks (of 512 bytes)
754 DupBlocks int64 // Blocks pointed from hard links
755 HLinks int64 // hard links
756 Words int64
757 Lines int64
758 Files int64
759 Dirs int64 // the num. of directories
760 SymLink int64
761 Flats int64 // the num. of flat files
762 MaxDepth int64
763 MaxNamlen int64 // max. name length
764 nextRepo time.Time
765 }
766 func showFusage(dir string, fusage *fileSum){
767     bsume := float64(((fusage.Blocks-fusage.DupBlocks)/2)*1024)/1000000.0
768     //bsumdup := float64((fusage.Blocks/2)*1024)/1000000.0
769
770     fmt.Printf("%v: %v files (%vd %vs %vh) %.6f MB (%.2f MBK)\n",
771         dir,
772         fusage.Files,
773         fusage.Dirs,
774         fusage.SymLink,
775         fusage.HLinks,
776         float64(fusage.Size)/1000000.0,bsume);
777 }
778 const (
779     S_IFMT = 0170000
780     S_IFCHR = 0020000
781     S_IFDIR = 0040000
782     S_IFREG = 0100000
783     S_IFLNK = 0120000
784     S_IFSOCK = 0140000
785 )
786 func cumPinfo(fsum *fileSum, path string, stater error, fstat syscall.Stat_t, argv[]string, verb bool)(*fileSum){
787     now := time.Now()
788     if time.Second <= now.Sub(fsum.nextRepo) {
789         if !fsum.nextRepo.IsZero(){
790             tstamp := now.Format(time.Stamp)
791             showFusage(tstamp, fsum)
792         }
793         fsum.nextRepo = now.Add(time.Second)
794     }
795     if stater != nil {
796         fsum.Err += 1
797         return fsum
798     }
799     fsum.Files += 1
800     if l < fstat.Nlink {
801         // must count only once...
802         // at least ignore ones in the same directory
803         //if finfo.Mode().IsRegular() {
804             if (fstat.Mode & S_IFMT) == S_IFREG {
805                 fsum.HLinks += 1
806                 fsum.DupBlocks += int64(fstat.Blocks)
807                 //fmt.Printf("---Dup HardLink %v %s\n", fstat.Nlink, path)
808             }
809         }
810         //fsum.Size += finfo.Size()
811         fsum.Size += fstat.Size
812         fsum.Blocks += int64(fstat.Blocks)
813         //if verb { fmt.Printf("(%8dBlk) %s", fstat.Blocks/2, path) }
814         if isin("-ls", argv){
815             //if verb { fmt.Printf("%4d %8d ", fstat.Blksize, fstat.Blocks) }
816             // fmt.Printf("%d\t", fstat.Blocks/2)
817         }
818         //if finfo.IsDir()
819         if (fstat.Mode & S_IFMT) == S_IFDIR {
820             fsum.Dirs += 1
821         }
822         //if (finfo.Mode() & os.ModeSymlink) != 0
823         if (fstat.Mode & S_IFMT) == S_IFLNK {
824             //if verb { fmt.Printf("symlink(%v,%s)\n", fstat.Mode, finfo.Name()) }
825             //{ fmt.Printf("symlink(%o,%s)\n", fstat.Mode, finfo.Name()) }
826             fsum.SymLink += 1
827         }
828         return fsum
829     }
830 func (gsh*GshContext)xxFindEntv(depth int, total *fileSum, dir string, dstat syscall.Stat_t, ei int, entv []string, npatv[]string, argv[]string)(*fileSum){
831     nols := isin("-grep", argv)
832     // sort entv
833     /*
834     if isin("-t", argv){
835         sort.Slice(filev, func(i,j int) bool {
836             return 0 < filev[i].ModTime().Sub(filev[j].ModTime())
837         })
838     }
839     */
840     /*
841     if isin("-u", argv){
842         sort.Slice(filev, func(i,j int) bool {
843             return 0 < filev[i].AccTime().Sub(filev[j].AccTime())
844         })
845     }
846     if isin("-U", argv){
847         sort.Slice(filev, func(i,j int) bool {
848             return 0 < filev[i].CreateTime().Sub(filev[j].CreateTime())
849         })
850     }
851     */
852     /*
853     if isin("-S", argv){
854         sort.Slice(filev, func(i,j int) bool {
855             return filev[j].Size() < filev[i].Size()
856         })
857     }
858     */
859     for _, filename := range entv {
860         for _, npat := range npatv {
861             match := true
862             if npat == "*" {
863                 match = true
864             }else{
865                 match, _ = filepath.Match(npat, filename)
866             }
867             path := dir + DIRSEP + filename
868             if !match {
869                 continue
870             }
871             var fstat syscall.Stat_t
872             stater := syscall.Lstat(path, &fstat)
873             if stater != nil {
874                 if !isin("-w", argv){fmt.Printf("ufind: %v\n", stater) }

```

```

875         continue;
876     }
877     if isin("-du",argv) && (fstat.Mode & S_IFMT) == S_IFDIR {
878         // should not show size of directory in "-du" mode ...
879     }else
880     if !nols && !isin("-s",argv) && (!isin("-du",argv) || isin("-a",argv)) {
881         if isin("-du",argv) {
882             fmt.Printf("%d\t",fstat.Blocks/2)
883         }
884         showFileInfo(path,argv)
885     }
886     if true { // && isin("-du",argv)
887         total = cumFinfo(total,path,staterr,fstat,argv,false)
888     }
889     /*
890     if isin("-wc",argv) {
891     }
892     */
893     if gsh.lastCheckSum.SumType != 0 {
894         gsh.xCksum(path,argv,gsh.lastCheckSum);
895     }
896     x := isinX("-grep",argv); // -grep will be convenient like -ls
897     if 0 < x && x+1 <= len(argv) { // -grep will be convenient like -ls
898         if IsRegFile(path){
899             found := gsh.xGrep(path,argv[x+1:])
900             if 0 < found {
901                 foundv := gsh.CmdCurrent.FoundFile
902                 if len(foundv) < 10 {
903                     gsh.CmdCurrent.FoundFile =
904                         append(gsh.CmdCurrent.FoundFile,path)
905                 }
906             }
907         }
908     }
909     if !isin("-r0",argv) { // -d 0 in du, -depth n in find
910         //total.Depth += 1
911         if (fstat.Mode & S_IFMT) == S_IFLNK {
912             continue
913         }
914         if dstat.Rdev != fstat.Rdev {
915             fmt.Printf("--I-- don't follow differnet device %v(%v) %v(%v)\n",
916                 dir,dstat.Rdev,path,fstat.Rdev)
917         }
918         if (fstat.Mode & S_IFMT) == S_IFDIR {
919             total = gsh.xxFind(depth+1,total,path,npatv,argv)
920         }
921     }
922 }
923 }
924 return total
925 }
926 func (gsh*GshContext)xxFind(depth int,total *fileSum,dir string,npatv[]string,argv[]string)(*fileSum){
927     nols := isin("-grep",argv)
928     dirfile,oerr := os.OpenFile(dir,os.O_RDONLY,0)
929     if oerr == nil {
930         //fmt.Printf("--I-- %v(%v)[%d]\n",dir,dirfile,dirfile.Fd())
931         defer dirfile.Close()
932     }else{
933     }
934 }
935 prev := *total
936 var dstat syscall.Stat_t
937 staterr := syscall.Lstat(dir,&dstat) // should be flstat
938 }
939 if staterr != nil {
940     if !isin("-w",argv){ fmt.Printf("ufind: %v\n",staterr) }
941     return total
942 }
943 //filev,err := ioutil.ReadDir(dir)
944 //_,err := ioutil.ReadDir(dir) // ReadDir() heavy and bad for huge directory
945 /*
946 if err != nil {
947     if !isin("-w",argv){ fmt.Printf("ufind: %v\n",err) }
948     return total
949 }
950 */
951 if depth == 0 {
952     total = cumFinfo(total,dir,staterr,dstat,argv,true)
953     if !nols && !isin("-s",argv) && (!isin("-du",argv) || isin("-a",argv)) {
954         showFileInfo(dir,argv)
955     }
956 }
957 // it is not a directory, just scan it and finish
958 }
959 for ei := 0; ; ei++ {
960     entv,rderr := dirfile.Readdirnames(8*1024)
961     if len(entv) == 0 || rderr != nil {
962         //if rderr != nil { fmt.Printf("[%d] len=%d (%v)\n",ei,len(entv),rderr) }
963         break
964     }
965     if 0 < ei {
966         fmt.Printf("--I-- xxFind[%d] %d large-dir: %s\n",ei,len(entv),dir)
967     }
968     total = gsh.xxFindEntv(depth,total,dir,dstat,ei,entv,npatv,argv)
969 }
970 if isin("-du",argv) {
971     // if in "du" mode
972     fmt.Printf("%d\t%s\n",total.Blocks-prev.Blocks)/2,dir)
973 }
974 return total
975 }
976 }
977 // {ufind|fu|ls} [Files] [// Names] [-- Expressions]
978 // Files is "." by default
979 // Names is "*" by default
980 // Expressions is "-print" by default for "ufind", or -du for "fu" command
981 func (gsh*GshContext)xFind(argv[]string){
982     if 0 < len(argv) && strBegins(argv[0],"?"){
983         showFound(gsh,argv)
984         return
985     }
986     if isin("-cksum",argv) || isin("-sum",argv) {
987         gsh.lastCheckSum = CheckSum{}
988         if isin("-sum",argv) && isin("-add",argv) {
989             gsh.lastCheckSum.SumType |= SUM_SUM64
990         }else
991         if isin("-sum",argv) && isin("-size",argv) {
992             gsh.lastCheckSum.SumType |= SUM_SIZE
993         }else
994         if isin("-sum",argv) && isin("-bsd",argv) {
995             gsh.lastCheckSum.SumType |= SUM_SUM16_BSD
996         }else
997         if isin("-sum",argv) && isin("-sysv",argv) {
998             gsh.lastCheckSum.SumType |= SUM_SUM16_SYSV
999         }else

```



```

1000     if isin("-sum",argv) {
1001         gsh.lastCheckSum.SumType |= SUM_SUM64
1002     }
1003     if isin("-unix",argv) {
1004         gsh.lastCheckSum.SumType |= SUM_UNIXFILE
1005         gsh.lastCheckSum.Crc32Table = *crc32.MakeTable(CRC32UNIX)
1006     }
1007     if isin("-ieee",argv){
1008         gsh.lastCheckSum.SumType |= SUM_CRCIEEE
1009         gsh.lastCheckSum.Crc32Table = *crc32.MakeTable(CRC32IEEE)
1010     }
1011     gsh.lastCheckSum.RusgAtStart = Getrusagev()
1012 }
1013 var total = fileSum()
1014 npats := []string{}
1015 for _,v := range argv {
1016     if 0 < len(v) && v[0] != '-' {
1017         npats = append(npats,v)
1018     }
1019     if v == "/" { break }
1020     if v == "--" { break }
1021     if v == "-grep" { break }
1022     if v == "-ls" { break }
1023 }
1024 if len(npats) == 0 {
1025     npats = []string{"*"}
1026 }
1027 cwd := "."
1028 // if to be fullpath ::: cwd, _ := os.Getwd()
1029 if len(npats) == 0 { npats = []string{"*"} }
1030 fusage := gsh.xxFind(0, &total, cwd, npats, argv)
1031 if gsh.lastCheckSum.SumType != 0 {
1032     var sumi uint64 = 0
1033     sum := &gsh.lastCheckSum
1034     if (sum.SumType & SUM_SIZE) != 0 {
1035         sumi = uint64(sum.Size)
1036     }
1037     if (sum.SumType & SUM_SUM64) != 0 {
1038         sumi = sum.Sum64
1039     }
1040     if (sum.SumType & SUM_SUM16_SYSV) != 0 {
1041         s := uint32(sum.Sum16)
1042         r := (s & 0xFFFF) + ((s & 0xFFFFFFFF) >> 16)
1043         s = (r & 0xFFFF) + (r >> 16)
1044         sum.Crc32Val = uint32(s)
1045         sumi = uint64(s)
1046     }
1047     if (sum.SumType & SUM_SUM16_BSD) != 0 {
1048         sum.Crc32Val = uint32(sum.Sum16)
1049         sumi = uint64(sum.Sum16)
1050     }
1051     if (sum.SumType & SUM_UNIXFILE) != 0 {
1052         sum.Crc32Val = byteCRC32end(sum.Crc32Val, uint64(sum.Size))
1053         sumi = uint64(byteCRC32end(sum.Crc32Val, uint64(sum.Size)))
1054     }
1055     if 1 < sum.Files {
1056         fmt.Printf("%v %v // %v / %v files, %v/file\r\n",
1057             sumi, sum.Size,
1058             abssize(sum.Size), sum.Files,
1059             abssize(sum.Size/sum.Files))
1060     }else{
1061         fmt.Printf("%v %v %v\n",
1062             sumi, sum.Size, npats[0])
1063     }
1064 }
1065 if !isin("-grep", argv) {
1066     showFusage("total", fusage)
1067 }
1068 if !isin("-s", argv){
1069     hits := len(gsh.CmdCurrent.FoundFile)
1070     if 0 < hits {
1071         fmt.Printf("--I-- %d files hits // can be refered with !&df\n",
1072             hits, len(gsh.CommandHistory))
1073     }
1074 }
1075 if gsh.lastCheckSum.SumType != 0 {
1076     if isin("-ru", argv) {
1077         sum := &gsh.lastCheckSum
1078         sum.Done = time.Now()
1079         gsh.lastCheckSum.RusgAtEnd = Getrusagev()
1080         elps := sum.Done.Sub(sum.Start)
1081         fmt.Printf("--cksum-size: %v (%v) / %v files, %v/file\r\n",
1082             sum.Size, abssize(sum.Size), sum.Files, abssize(sum.Size/sum.Files))
1083         nanos := int64(elps)
1084         fmt.Printf("--cksum-time: %v/total, %v/file, %.1f files/s, %v\r\n",
1085             abstime(nanos),
1086             abstime(nanos/sum.Files),
1087             (float64(sum.Files)*1000000000.0)/float64(nanos),
1088             abbspeed(sum.Size, nanos))
1089         diff := RusageSubv(sum.RusgAtEnd, sum.RusgAtStart)
1090         fmt.Printf("--cksum-rusg: %v\n", sRusagef("", argv, diff))
1091     }
1092 }
1093 return
1094 }
1095
1096 func showFiles(files []string){
1097     sp := ""
1098     for i, file := range files {
1099         if 0 < i { sp = " " } else { sp = "" }
1100         fmt.Printf(sp+"%s", escapeWhitesP(file))
1101     }
1102 }
1103 func showFound(gshCtx *GshContext, argv []string){
1104     for i, v := range gshCtx.CommandHistory {
1105         if 0 < len(v.FoundFile) {
1106             fmt.Printf("%d (%d) ", i, len(v.FoundFile))
1107             if isin("-ls", argv){
1108                 fmt.Printf("\n")
1109                 for _, file := range v.FoundFile {
1110                     fmt.Printf(" ") //sub number?
1111                     showFileInfo(file, argv)
1112                 }
1113             }else{
1114                 showFiles(v.FoundFile)
1115                 fmt.Printf("\n")
1116             }
1117         }
1118     }
1119 }
1120
1121 func showMatchFile(filev []os.FileInfo, npat, dir string, argv []string)(string, bool){
1122     fname := ""
1123     found := false
1124     for _, v := range filev {

```

```

1125     match, _ := filepath.Match(npat, (v.Name()))
1126     if match {
1127         fname = v.Name()
1128         found = true
1129         //fmt.Printf("[%d] %s\n", i, v.Name())
1130         showIfExecutable(fname, dir, argv)
1131     }
1132 }
1133 return fname, found
1134 }
1135 func showIfExecutable(name, dir string, argv []string) (ffullpath string, ffound bool) {
1136     var fullpath string
1137     if strBegins(name, DIRSEP) {
1138         fullpath = name
1139     } else {
1140         fullpath = dir + DIRSEP + name
1141     }
1142     fi, err := os.Stat(fullpath)
1143     if err != nil {
1144         fullpath = dir + DIRSEP + name + ".go"
1145         fi, err = os.Stat(fullpath)
1146     }
1147     if err == nil {
1148         fm := fi.Mode()
1149         if fm.IsRegular() {
1150             // R_OK=4, W_OK=2, X_OK=1, F_OK=0
1151             if syscall.Access(fullpath, 5) == nil {
1152                 ffullpath = fullpath
1153                 ffound = true
1154                 if !isin("-s", argv) {
1155                     showFileInfo(fullpath, argv)
1156                 }
1157             }
1158         }
1159     }
1160     return ffullpath, ffound
1161 }
1162 func which(list string, argv []string) (fullpathv []string, itis bool) {
1163     if len(argv) <= 1 {
1164         fmt.Printf("Usage: which comand [-s] [-a] [-ls]\n")
1165         return []string{"", false}
1166     }
1167     path := argv[1]
1168     if strBegins(path, "/") {
1169         // should check if executable?
1170         _, exOK := showIfExecutable(path, "/", argv)
1171         fmt.Printf("--D-- %v exOK=%v\n", path, exOK)
1172         return []string{path}, exOK
1173     }
1174     pathenv, efound := os.LookupEnv(list)
1175     if !efound {
1176         fmt.Printf("--E-- which: no \"%s\" environment\n", list)
1177         return []string{"", false}
1178     }
1179     showall := isin("-a", argv) || 0 <= strings.Index(path, "*")
1180     dirv := strings.Split(pathenv, PATHSEP)
1181     ffound := false
1182     ffullpath := path
1183     for _, dir := range dirv {
1184         if 0 <= strings.Index(path, "*") { // by wild-card
1185             list, _ := ioutil.ReadDir(dir)
1186             ffullpath, ffound = showMatchFile(list, path, dir, argv)
1187         } else {
1188             ffullpath, ffound = showIfExecutable(path, dir, argv)
1189         }
1190         //if ffound && !isin("-a", argv) {
1191         if ffound && !showall {
1192             break;
1193         }
1194     }
1195     return []string{ffullpath}, ffound
1196 }
1197
1198 func stripLeadingWSParg(argv []string) ([]string) {
1199     for i, 0 < len(argv); {
1200         if len(argv[0]) == 0 {
1201             argv = argv[1:]
1202         } else {
1203             break
1204         }
1205     }
1206     return argv
1207 }
1208 func xEval(argv []string, nlend bool) {
1209     argv = stripLeadingWSParg(argv)
1210     if len(argv) == 0 {
1211         fmt.Printf("eval [%sformat] [Go-expression]\n")
1212         return
1213     }
1214     pfmt := "%v"
1215     if argv[0][0] == '$' {
1216         pfmt = argv[0]
1217         argv = argv[1:]
1218     }
1219     if len(argv) == 0 {
1220         return
1221     }
1222     gocode := strings.Join(argv, " ");
1223     //fmt.Printf("eval [%v] [%v]\n", pfmt, gocode)
1224     fset := token.NewFileSet()
1225     rval, _ := types.Eval(fset, nil, token.NoPos, gocode)
1226     fmt.Printf(pfmt, rval.Value)
1227     if nlend { fmt.Printf("\n") }
1228 }
1229
1230 func getval(name string) (found bool, val int) {
1231     /* should expand the name here */
1232     if name == "gsh.pid" {
1233         return true, os.Getpid()
1234     } else {
1235         if name == "gsh.ppid" {
1236             return true, os.Getppid()
1237         }
1238     }
1239     return false, 0
1240 }
1241
1242 func echo(argv []string, nlend bool) {
1243     for ai := 1; ai < len(argv); ai++ {
1244         if 1 < ai {
1245             fmt.Printf(" ");
1246         }
1247         arg := argv[ai]
1248         found, val := getval(arg)
1249         if found {
1250             fmt.Printf("%d", val)
1251         }
1252     }
1253     if nlend {
1254         fmt.Printf("\n")
1255     }
1256 }

```

```

1250     }else{
1251         fmt.Printf("%s",arg)
1252     }
1253 }
1254 if nlen {
1255     fmt.Printf("\n");
1256 }
1257 }
1258 }
1259 func resfile() string {
1260     return "gsh.tmp"
1261 }
1262 //var resF *File
1263 func resmap() {
1264     //_, err := os.OpenFile(resfile(), os.O_RDWR|os.O_CREATE, os.ModeAppend)
1265     // https://deveppaper.com/solution-to-golang-bad-file-descriptor-problem/
1266     _, err := os.OpenFile(resfile(), os.O_RDWR|os.O_CREATE, 0600)
1267     if err != nil {
1268         fmt.Printf("refF could not open: %s\n",err)
1269     }else{
1270         fmt.Printf("refF opened\n")
1271     }
1272 }
1273 }
1274 // @@2020-0821
1275 func gshScanArg(str string,strip int)(argv []string){
1276     var si = 0
1277     var sb = 0
1278     var inBracket = 0
1279     var argl = make([]byte,LINESIZE)
1280     var ax = 0
1281     debug := false
1282 }
1283 for ; si < len(str); si++ {
1284     if str[si] != ' ' {
1285         break
1286     }
1287 }
1288 sb = si
1289 for ; si < len(str); si++ {
1290     if sb <= si {
1291         if debug {
1292             fmt.Printf("--Da- +%d %2d-%2d %s ... %s\n",
1293                 inBracket,sb,si,argl[0:ax],str[si:])
1294         }
1295     }
1296     ch := str[si]
1297     if ch == '{' {
1298         inBracket += 1
1299         if 0 < strip && inBracket <= strip {
1300             //fmt.Printf("stripLEV %d <= %d?\n",inBracket,strip)
1301             continue
1302         }
1303     }
1304     if 0 < inBracket {
1305         if ch == '}' {
1306             inBracket -= 1
1307             if 0 < strip && inBracket < strip {
1308                 //fmt.Printf("stripLEV %d < %d?\n",inBracket,strip)
1309                 continue
1310             }
1311         }
1312         argl[ax] = ch
1313         ax += 1
1314         continue
1315     }
1316     if str[si] == ' ' {
1317         argv = append(argv,string(argl[0:ax]))
1318         if debug {
1319             fmt.Printf("--Da- [%v][%v-%v] %s ... %s\n",
1320                 -1+len(argv),sb,si,str[sb:si],string(str[si:]))
1321         }
1322         sb = si+1
1323         ax = 0
1324         continue
1325     }
1326     argl[ax] = ch
1327     ax += 1
1328 }
1329 if sb < si {
1330     argv = append(argv,string(argl[0:ax]))
1331     if debug {
1332         fmt.Printf("--Da- [%v][%v-%v] %s ... %s\n",
1333             -1+len(argv),sb,si,string(argl[0:ax]),string(str[si:]))
1334     }
1335 }
1336 if debug {
1337     fmt.Printf("--Da- %d [%s] => [%d]%v\n",strip,str,len(argv),argv)
1338 }
1339 return argv
1340 }
1341 }
1342 // should get stderr (into tmpfile ?) and return
1343 func (gsh*GshContext)Popen(name,mode string)(pin*os.File,pout*os.File,err bool){
1344     var pv = []int{-1,-1}
1345     syscall.Pipe(pv)
1346 }
1347 xarg := gshScanArg(name,1)
1348 name = strings.Join(xarg," ")
1349 }
1350 pin = os.NewFile(uintptr(pv[0]),"StdoutOf-"+name+"")
1351 pout = os.NewFile(uintptr(pv[1]),"StdinOf-"+name+"")
1352 fdix := 0
1353 dir := "?"
1354 if mode == "r" {
1355     dir = "<"
1356     fdix = 1 // read from the stdout of the process
1357 }else{
1358     dir = ">"
1359     fdix = 0 // write to the stdin of the process
1360 }
1361 gshPA := gsh.gshPA
1362 savfd := gshPA.Files[fdix]
1363 }
1364 var fd uintptr = 0
1365 if mode == "r" {
1366     fd = pout.Fd()
1367     gshPA.Files[fdix] = pout.Fd()
1368 }else{
1369     fd = pin.Fd()
1370     gshPA.Files[fdix] = pin.Fd()
1371 }
1372 // should do this by Goroutine?
1373 if false {
1374     fmt.Printf("--Ip- Opened fd[%v] %s %v\n",fd,dir,name)

```

```

1375         fmt.Printf("--RED1 [%d,%d,%d]->[%d,%d,%d]\n",
1376             os.Stdin.Fd(),os.Stdout.Fd(),os.Stderr.Fd(),
1377             pin.Fd(),pout.Fd(),pout.Fd())
1378     }
1379     savi := os.Stdin
1380     savo := os.Stdout
1381     save := os.Stderr
1382     os.Stdin = pin
1383     os.Stdout = pout
1384     os.Stderr = pout
1385     gsh.BackGround = true
1386     gsh.gshellh(name)
1387     gsh.BackGround = false
1388     os.Stdin = savi
1389     os.Stdout = savo
1390     os.Stderr = save
1391
1392     gshPA.Files[fdix] = savfd
1393     return pin,pout,false
1394 }
1395
1396 // <a name="ex-commands">External commands</a>
1397 func (gsh*GshContext)excommand(exec bool, argv []string) (notf bool,exit bool) {
1398     if gsh.CmdTrace { fmt.Printf("--I-- excommand[%v](%v)\n",exec,argv) }
1399
1400     gshPA := gsh.gshPA
1401     fullpathv, itis := which("PATH",[]string{"which",argv[0],"-s"})
1402     if itis == false {
1403         return true,false
1404     }
1405     fullpath := fullpathv[0]
1406     argv = unescapeWhiteSPV(argv)
1407     if 0 < strings.Index(fullpath,".go") {
1408         nargv := argv // []string{}
1409         gofullpathv, itis := which("PATH",[]string{"which","go","-s"})
1410         if itis == false {
1411             fmt.Printf("--F-- Go not found\n")
1412             return false,true
1413         }
1414         gofullpath := gofullpathv[0]
1415         nargv = []string{ gofullpath, "run", fullpath }
1416         fmt.Printf("--I-- %s %s %s %s\n",gofullpath,
1417             nargv[0],nargv[1],nargv[2])
1418         if exec {
1419             syscall.Exec(gofullpath,nargv,os.Environ())
1420         }else{
1421             pid, _ := syscall.ForkExec(gofullpath,nargv,&gshPA)
1422             if gsh.BackGround {
1423                 fmt.Fprintf(stderr,"--Ip- in Background pid[%d]%d(%v)\n",pid,len(argv),nargv)
1424                 gsh.BackGroundJobs = append(gsh.BackGroundJobs,pid)
1425             }else{
1426                 rusage := syscall.Rusage {}
1427                 syscall.Wait4(pid,nil,0,&rusage)
1428                 gsh.LastRusage = rusage
1429                 gsh.CmdCurrent.Rusagev[1] = rusage
1430             }
1431         }
1432     }else{
1433         if exec {
1434             syscall.Exec(fullpath,argv,os.Environ())
1435         }else{
1436             pid, _ := syscall.ForkExec(fullpath,argv,&gshPA)
1437             //fmt.Printf("[%d]\n",pid); // '&' to be background
1438             if gsh.BackGround {
1439                 fmt.Fprintf(stderr,"--Ip- in Background pid[%d]%d(%v)\n",pid,len(argv),argv)
1440                 gsh.BackGroundJobs = append(gsh.BackGroundJobs,pid)
1441             }else{
1442                 rusage := syscall.Rusage {}
1443                 syscall.Wait4(pid,nil,0,&rusage);
1444                 gsh.LastRusage = rusage
1445                 gsh.CmdCurrent.Rusagev[1] = rusage
1446             }
1447         }
1448     }
1449     return false,false
1450 }
1451
1452 // <a name="builtin">Builtin Commands</a>
1453 func (gshCtx *GshContext) sleep(argv []string) {
1454     if len(argv) < 2 {
1455         fmt.Printf("Sleep 100ms, 100us, 100ns, ...)\n")
1456         return
1457     }
1458     duration := argv[1];
1459     d, err := time.ParseDuration(duration)
1460     if err != nil {
1461         d, err = time.ParseDuration(duration+"s")
1462         if err != nil {
1463             fmt.Printf("duration ? %s (%s)\n",duration,err)
1464             return
1465         }
1466     }
1467     //fmt.Printf("Sleep %v\n",duration)
1468     time.Sleep(d)
1469     if 0 < len(argv[2:]) {
1470         gshCtx.gshellv(argv[2:])
1471     }
1472 }
1473 func (gshCtx *GshContext)repeat(argv []string) {
1474     if len(argv) < 2 {
1475         return
1476     }
1477     start0 := time.Now()
1478     for ri, _ := strconv.Atoi(argv[1]); 0 < ri; ri-- {
1479         if 0 < len(argv[2:]) {
1480             //start := time.Now()
1481             gshCtx.gshellv(argv[2:])
1482             end := time.Now()
1483             elps := end.Sub(start0);
1484             if( 1000000000 < elps ){
1485                 fmt.Printf("(repeat#%d %v)\n",ri,elps);
1486             }
1487         }
1488     }
1489 }
1490
1491 func (gshCtx *GshContext)gen(argv []string) {
1492     gshPA := gshCtx.gshPA
1493     if len(argv) < 2 {
1494         fmt.Printf("Usage: %s N\n",argv[0])
1495         return
1496     }
1497     // should br repeated by "repeat" command
1498     count, _ := strconv.Atoi(argv[1])
1499     fd := gshPA.Files[1] // Stdout

```

```

1500 file := os.NewFile(fd, "internalStdOut")
1501 fmt.Printf("--In- Gen. Count=%d to [%d]\n", count, file.Fd())
1502 //buf := []byte{}
1503 outdata := "0123 5678 0123 5678 0123 5678 0123 5678\r"
1504 for gi := 0; gi < count; gi++ {
1505     file.WriteString(outdata)
1506 }
1507 //file.WriteString("\n")
1508 fmt.Printf("\n(%d B)\n", count*len(outdata));
1509 //file.Close()
1510 }
1511 }
1512 // <a name="rexec">Remote Execution</a> // 2020-0820
1513 func Elapsed(from time.Time)(string){
1514     elps := time.Now().Sub(from)
1515     if 1000000000 < elps {
1516         return fmt.Sprintf("[%5d.%02ds]", elps/1000000000, (elps%1000000000)/1000000)
1517     }else
1518     if 1000000 < elps {
1519         return fmt.Sprintf("[%3d.%03dms]", elps/1000000, (elps%1000000)/1000)
1520     }else{
1521         return fmt.Sprintf("[%3d.%03dus]", elps/1000, (elps%1000))
1522     }
1523 }
1524 func abftime(nanos int64)(string){
1525     if 1000000000 < nanos {
1526         return fmt.Sprintf("%d.%02ds", nanos/1000000000, (nanos%1000000000)/1000000)
1527     }else
1528     if 1000000 < nanos {
1529         return fmt.Sprintf("%d.%03dms", nanos/1000000, (nanos%1000000)/1000)
1530     }else{
1531         return fmt.Sprintf("%d.%03dus", nanos/1000, (nanos%1000))
1532     }
1533 }
1534 func absbsize(size int64)(string){
1535     fsize := float64(size)
1536     if 1024*1024*1024 < size {
1537         return fmt.Sprintf("%.2fGiB", fsize/(1024*1024*1024))
1538     }else
1539     if 1024*1024 < size {
1540         return fmt.Sprintf("%.3fMiB", fsize/(1024*1024))
1541     }else{
1542         return fmt.Sprintf("%.3fKiB", fsize/1024)
1543     }
1544 }
1545 func absize(size int64)(string){
1546     fsize := float64(size)
1547     if 1024*1024*1024 < size {
1548         return fmt.Sprintf("%.2fGiB", fsize/(1024*1024*1024))
1549     }else
1550     if 1024*1024 < size {
1551         return fmt.Sprintf("%.3fMiB", fsize/(1024*1024))
1552     }else{
1553         return fmt.Sprintf("%.3fKiB", fsize/1024)
1554     }
1555 }
1556 func abbspd(totalB int64, ns int64)(string){
1557     MBs := (float64(totalB)/1000000) / (float64(ns)/1000000000)
1558     if 1000 <= MBs {
1559         return fmt.Sprintf("%.3fGB/s", MBs/1000)
1560     }
1561     if 1 <= MBs {
1562         return fmt.Sprintf("%.3fMB/s", MBs)
1563     }else{
1564         return fmt.Sprintf("%.3fKB/s", MBs*1000)
1565     }
1566 }
1567 func abspsd(totalB int64, ns time.Duration)(string){
1568     MBs := (float64(totalB)/1000000) / (float64(ns)/1000000000)
1569     if 1000 <= MBs {
1570         return fmt.Sprintf("%.3fGBps", MBs/1000)
1571     }
1572     if 1 <= MBs {
1573         return fmt.Sprintf("%.3fMBps", MBs)
1574     }else{
1575         return fmt.Sprintf("%.3fKBps", MBs*1000)
1576     }
1577 }
1578 func fileRelay(what string, in*os.File, out*os.File, size int64, bsiz int)(wcount int64){
1579     Start := time.Now()
1580     buff := make([]byte, bsiz)
1581     var total int64 = 0
1582     var rem int64 = size
1583     nio := 0
1584     Prev := time.Now()
1585     var PrevSize int64 = 0
1586
1587     fmt.Printf(Elapsed(Start)+"--In- X: %s (%v/%v/%v) START\n",
1588         what, absize(total), size, nio)
1589
1590     for i:= 0; ; i++ {
1591         var len = bsiz
1592         if int(rem) < len {
1593             len = int(rem)
1594         }
1595         Now := time.Now()
1596         Elps := Now.Sub(Prev);
1597         if 1000000000 < Now.Sub(Prev) {
1598             fmt.Printf(Elapsed(Start)+"--In- X: %s (%v/%v/%v) %s\n",
1599                 what, absize(total), size, nio,
1600                 abspsd((total-PrevSize), Elps))
1601             Prev = Now;
1602             PrevSize = total
1603         }
1604         rlen := len
1605         if in != nil {
1606             // should watch the disconnection of out
1607             rcc, err := in.Read(buff[0:rlen])
1608             if err != nil {
1609                 fmt.Printf(Elapsed(Start)+"--En- X: %s read(%v,%v)<%v\n",
1610                     what, rcc, err, in.Name())
1611                 break
1612             }
1613             rlen = rcc
1614             if string(buff[0:10]) == "(SoftEOF " {
1615                 var ecc int64 = 0
1616                 fmt.Sscanf(string(buff), "(SoftEOF %v", &ecc)
1617                 fmt.Printf(Elapsed(Start)+"--En- X: %s Recv ((SoftEOF %v))&v\n",
1618                     what, ecc, total)
1619                 if ecc == total {
1620                     break
1621                 }
1622             }
1623         }
1624     }

```

```

1625     wlen := rlen
1626     if out != nil {
1627         wcc,err := out.Write(buff[0:rlen])
1628         if err != nil {
1629             fmt.Printf(Elapsed(Start)+"--En- X: %s write(%v,%v)>%v\n",
1630                 what,wcc,err,out.Name())
1631             break
1632         }
1633         wlen = wcc
1634     }
1635     if wlen < rlen {
1636         fmt.Printf(Elapsed(Start)+"--En- X: %s incomplete write (%v/%v)\n",
1637             what,wlen,rlen)
1638         break;
1639     }
1640
1641     nio += 1
1642     total += int64(rlen)
1643     rem -= int64(rlen)
1644     if rem <= 0 {
1645         break
1646     }
1647 }
1648 Done := time.Now()
1649 Elps := float64(Done.Sub(Start))/1000000000 //Seconds
1650 TotalMB := float64(total)/1000000 //MB
1651 MBps := TotalMB / Elps
1652 fmt.Printf(Elapsed(Start)+"--In- X: %s (%v/%v/%v) %v %v.3fMB/s\n",
1653     what,total,size,nio,absize(total),MBps)
1654 return total
1655 }
1656 func tcpPush(clnt *os.File){
1657     // shrink socket buffer and recover
1658     usleep(100);
1659 }
1660 func (gsh*GshContext)RexecServer(argv[]string){
1661     debug := true
1662     Start0 := time.Now()
1663     Start := Start0
1664     // if local == "" { local = "0.0.0.0:9999" }
1665     local := "0.0.0.0:9999"
1666
1667     if 0 < len(argv) {
1668         if argv[0] == "-s" {
1669             debug = false
1670             argv = argv[1:]
1671         }
1672     }
1673     if 0 < len(argv) {
1674         argv = argv[1:]
1675     }
1676     port, err := net.ResolveTCPAddr("tcp",local);
1677     if err != nil {
1678         fmt.Printf("--En- S: Address error: %s (%s)\n",local,err)
1679         return
1680     }
1681     fmt.Printf(Elapsed(Start)+"--In- S: Listening at %s...\n",local);
1682     sconn, err := net.ListenTCP("tcp", port)
1683     if err != nil {
1684         fmt.Printf(Elapsed(Start)+"--En- S: Listen error: %s (%s)\n",local,err)
1685         return
1686     }
1687
1688     reqbuf := make([]byte,LINESIZE)
1689     res := ""
1690     for {
1691         fmt.Printf(Elapsed(Start0)+"--In- S: Listening at %s...\n",local);
1692         aconn, err := sconn.AcceptTCP()
1693         Start = time.Now()
1694         if err != nil {
1695             fmt.Printf(Elapsed(Start)+"--En- S: Accept error: %s (%s)\n",local,err)
1696             return
1697         }
1698         clnt, _ := aconn.File()
1699         fd := Clnt.Fd()
1700         ar := aconn.RemoteAddr()
1701         if debug { fmt.Printf(Elapsed(Start0)+"--In- S: Accepted TCP at %s [%d] <- %v\n",
1702             local,fd,ar) }
1703         res = fmt.Sprintf("220 GShell/%s Server\r\n",VERSION)
1704         fmt.Fprintf(clnt,"%s",res)
1705         if debug { fmt.Printf(Elapsed(Start)+"--In- S: %s",res) }
1706         count, err := clnt.Read(reqbuf)
1707         if err != nil {
1708             fmt.Printf(Elapsed(Start)+"--En- C: (%v %v) %v",
1709                 count,err,string(reqbuf))
1710         }
1711         req := string(reqbuf[:count])
1712         if debug { fmt.Printf(Elapsed(Start)+"--In- C: %v",string(req)) }
1713         reqv := strings.Split(string(req),"r")
1714         cmdv := gshScanArg(reqv[0],0)
1715         //cmdv := strings.Split(reqv[0]," ")
1716         switch cmdv[0] {
1717             case "HELO":
1718                 res = fmt.Sprintf("250 %v",req)
1719             case "GET":
1720                 // download {remotefile|-zN} [localfile]
1721                 var dsize int64 = 32*1024*1024
1722                 var bsize int = 64*1024
1723                 var fname string = ""
1724                 var in *os.File = nil
1725                 var pseudoEOF = false
1726                 if 1 < len(cmdv) {
1727                     fname = cmdv[1]
1728                     if strBegins(fname,"-z") {
1729                         fmt.Sscanf(fname[2:], "%d",&dsize)
1730                     }else
1731                     if strBegins(fname,"{") {
1732                         xin,xout,err := gsh.Popen(fname,"r")
1733                         if err {
1734                             }else{
1735                                 xout.Close()
1736                                 defer xin.Close()
1737                                 in = xin
1738                                 dsize = MaxStreamSize
1739                                 pseudoEOF = true
1740                             }
1741                     }else{
1742                         xin,err := os.Open(fname)
1743                         if err != nil {
1744                             fmt.Printf("--En- GET (%v)\n",err)
1745                         }else{
1746                             defer xin.Close()
1747                             in = xin
1748                             fi,_ := xin.Stat()
1749                             dsize = fi.Size()

```

```

1750     }
1751     }
1752     }
1753     //fmt.Printf(Elapsed(Start)+"--In- GET %v:%v\n",dsize,bsize)
1754     res = fmt.Sprintf("200 %v\r\n",dsize)
1755     fmt.Fprintf(clnt, "%v",res)
1756     tcpPush(clnt); // should be separated as line in receiver
1757     fmt.Printf(Elapsed(Start)+"--In- S: %v",res)
1758     wcount := fileRelay("SendGET",in,clnt,dsize,bsize)
1759     if pseudoEOF {
1760         in.Close() // pipe from the command
1761         // show end of stream data (its size) by OOB?
1762         SoftEOF := fmt.Sprintf("({SoftEOF %v})",wcount)
1763         fmt.Printf(Elapsed(Start)+"--In- S: Send %v\n",SoftEOF)
1764
1765         tcpPush(clnt); // to let SoftEOF data appear at the top of received data
1766         fmt.Fprintf(clnt, "%v\r\n",SoftEOF)
1767         tcpPush(clnt); // to let SoftEOF alone in a packet (separate with 200 OK)
1768         // with client generated random?
1769         //fmt.Printf("--In- L: close %v (%v)\n",in.Fd(),in.Name())
1770     }
1771     res = fmt.Sprintf("200 GET done\r\n")
1772     case "PUT":
1773         // upload {srcfile|-zN} [dstfile]
1774         var dsize int64 = 32*1024*1024
1775         var bsize int = 64*1024
1776         var fname string = ""
1777         var out *os.File = nil
1778         if 1 < len(cmdv) { // localfile
1779             fmt.Sscanf(cmdv[1],"%d",&dsize)
1780         }
1781         if 2 < len(cmdv) {
1782             fname = cmdv[2]
1783             if fname == "-" {
1784                 // nul dev
1785             }else
1786             if strBegins(fname,".") {
1787                 xin,xout,err := gsh.Popen(fname,"w")
1788                 if err {
1789                     }else{
1790                         xin.Close()
1791                         defer xout.Close()
1792                         out = xout
1793                     }
1794             }else{
1795                 // should write to temporary file
1796                 // should suppress ^C on tty
1797                 xout,err := os.OpenFile(fname,os.O_CREATE|os.O_RDWR|os.O_TRUNC,0600)
1798                 //fmt.Printf("--In- S: open(%v) out(%v) err(%v)\n",fname,xout,err)
1799                 if err != nil {
1800                     fmt.Printf("--En- PUT (%v)\n",err)
1801                 }else{
1802                     out = xout
1803                 }
1804             }
1805             fmt.Printf(Elapsed(Start)+"--In- L: open(%v,w) %v (%v)\n",
1806                 fname,local,err)
1807         }
1808         fmt.Printf(Elapsed(Start)+"--In- PUT %v (%v)\n",dsize,bsize)
1809         fmt.Printf(Elapsed(Start)+"--In- S: 200 %v OK\r\n",dsize)
1810         fmt.Fprintf(clnt,"200 %v OK\r\n",dsize)
1811         fileRelay("RecvPUT",clnt,out,dsize,bsize)
1812         res = fmt.Sprintf("200 PUT done\r\n")
1813     default:
1814         res = fmt.Sprintf("400 What? %v",req)
1815     }
1816     swcc,serr := clnt.Write([]byte(res))
1817     if serr != nil {
1818         fmt.Printf(Elapsed(Start)+"--In- S: (wc=%v er=%v) %v",swcc,serr,res)
1819     }else{
1820         fmt.Printf(Elapsed(Start)+"--In- S: %v",res)
1821     }
1822     aconn.Close();
1823     clnt.Close();
1824 }
1825 sconn.Close();
1826 }
1827 func (gsh*GshContext)RexecClient(argv []string)(int,string){
1828     debug := true
1829     Start := time.Now()
1830     if len(argv) == 1 {
1831         return -1,"EmptyARG"
1832     }
1833     argv = argv[1:]
1834     if argv[0] == "-serv" {
1835         gsh.RexecServer(argv[1:])
1836         return 0,"Server"
1837     }
1838     remote := "0.0.0.0:9999"
1839     if argv[0][0] == '@' {
1840         remote = argv[0][1:]
1841         argv = argv[1:]
1842     }
1843     if argv[0] == "-s" {
1844         debug = false
1845         argv = argv[1:]
1846     }
1847     dport, err := net.ResolveTCPAddr("tcp",remote);
1848     if err != nil {
1849         fmt.Printf(Elapsed(Start)+"Address error: %s (%s)\n",remote,err)
1850         return -1,"AddressError"
1851     }
1852     fmt.Printf(Elapsed(Start)+"--In- C: Connecting to %s\n",remote)
1853     serv, err := net.DialTCP("tcp",nil,dport)
1854     if err != nil {
1855         fmt.Printf(Elapsed(Start)+"Connection error: %s (%s)\n",remote,err)
1856         return -1,"CannotConnect"
1857     }
1858     if debug {
1859         al := serv.LocalAddr()
1860         fmt.Printf(Elapsed(Start)+"--In- C: Connected to %v <- %v\n",remote,al)
1861     }
1862 }
1863 req := ""
1864 res := make([]byte,LINESIZE)
1865 count,err := serv.Read(res)
1866 if err != nil {
1867     fmt.Printf("--En- S: (%3d,%v) %v",count,err,string(res))
1868 }
1869 if debug { fmt.Printf(Elapsed(Start)+"--In- S: %v",string(res)) }
1870 }
1871 if argv[0] == "GET" {
1872     savPA := gsh.gshPA
1873     var bsize int = 64*1024
1874     req = fmt.Sprintf("%v\r\n",strings.Join(argv, " "))

```

```

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

```



```

2000 //fmt.Printf("--D-- Client: %v (%v)",rcode,stat)
2001 return rcode,ress
2002 }
2003
2004 // <a name="remote-sh">Remote Shell</a>
2005 // gcp file [...] { [host]:[port]:[dir] | dir } // -p | -no-p
2006 func (gsh*GshContext)FileCopy(argv []string){
2007     var host = ""
2008     var port = ""
2009     var upload = false
2010     var download = false
2011     var xargv = []string{"rex-gcp"}
2012     var srcv = []string{}
2013     var dstv = []string{}
2014     argv = argv[1:]
2015
2016     for _,v := range argv {
2017         /*
2018         if v[0] == '-' { // might be a pseudo file (generated date)
2019             continue
2020         }
2021         */
2022         obj := strings.Split(v,":")
2023         //fmt.Printf("%d %v %v\n",len(obj),v,obj)
2024         if 1 < len(obj) {
2025             host = obj[0]
2026             file := ""
2027             if 0 < len(host) {
2028                 gsh.LastServer.host = host
2029             }else{
2030                 host = gsh.LastServer.host
2031                 port = gsh.LastServer.port
2032             }
2033             if 2 < len(obj) {
2034                 port = obj[1]
2035                 if 0 < len(port) {
2036                     gsh.LastServer.port = port
2037                 }else{
2038                     port = gsh.LastServer.port
2039                 }
2040                 file = obj[2]
2041             }else{
2042                 file = obj[1]
2043             }
2044             if len(srcv) == 0 {
2045                 download = true
2046                 srcv = append(srcv,file)
2047                 continue
2048             }
2049             upload = true
2050             dstv = append(dstv,file)
2051             continue
2052         }
2053         /*
2054         idx := strings.Index(v,":")
2055         if 0 <= idx {
2056             remote = v[0:idx]
2057             if len(srcv) == 0 {
2058                 download = true
2059                 srcv = append(srcv,v[idx+1:])
2060                 continue
2061             }
2062             upload = true
2063             dstv = append(dstv,v[idx+1:])
2064             continue
2065         }
2066         */
2067         if download {
2068             dstv = append(dstv,v)
2069         }else{
2070             srcv = append(srcv,v)
2071         }
2072     }
2073     hostport := "@" + host + ":" + port
2074     if upload {
2075         if host != "" { xargv = append(xargv,hostport) }
2076         xargv = append(xargv,"PUT")
2077         xargv = append(xargv,srcv[0]...)
2078         xargv = append(xargv,dstv[0]...)
2079         //fmt.Printf("--I-- FileCopy PUT gsh://s/%v < %v // %v\n",hostport,dstv,srcv)
2080         fmt.Printf("--I-- FileCopy PUT gsh://s/%v < %v\n",hostport,dstv,srcv)
2081         gsh.RexecClient(xargv)
2082     }else
2083     if download {
2084         if host != "" { xargv = append(xargv,hostport) }
2085         xargv = append(xargv,"GET")
2086         xargv = append(xargv,srcv[0]...)
2087         xargv = append(xargv,dstv[0]...)
2088         //fmt.Printf("--I-- FileCopy GET gsh://v/%v > %v // %v\n",hostport,srcv,dstv)
2089         fmt.Printf("--I-- FileCopy GET gsh://v/%v > %v\n",hostport,srcv,dstv)
2090         gsh.RexecClient(xargv)
2091     }else{
2092     }
2093 }
2094
2095 // target
2096 func (gsh*GshContext)Trelpath(rloc string)(string){
2097     cwd, _ := os.Getwd()
2098     os.Chdir(gsh.RWD)
2099     os.Chdir(rloc)
2100     twd, _ := os.Getwd()
2101     os.Chdir(cwd)
2102
2103     tpath := twd + "/" + rloc
2104     return tpath
2105 }
2106 // join to rnote GShell - [user@]host[:port] or cd host[:port]:path
2107 func (gsh*GshContext)Rjoin(argv []string){
2108     if len(argv) <= 1 {
2109         fmt.Printf("--I-- current server = %v\n",gsh.RSERV)
2110         return
2111     }
2112     serv := argv[1]
2113     servv := strings.Split(serv,":")
2114     if 1 <= len(servv) {
2115         if servv[0] == "lo" {
2116             servv[0] = "localhost"
2117         }
2118     }
2119     switch len(servv) {
2120     case 1:
2121         //if strings.Index(serv,":") < 0 {
2122             serv = servv[0] + ":" + fmt.Sprintf("%d",GSH_PORT)
2123         //}
2124     case 2: // host:port

```

```

2125     serv = strings.Join(servv,":")
2126 }
2127 xargv := []string{"rex-join","@"+serv,"HELO"}
2128 rcode,stat := gsh.RexecClient(xargv)
2129 if (rcode / 100) == 2 {
2130     fmt.Printf("--I-- OK Joined (%v) %v\n",rcode,stat)
2131     gsh.RSERV = serv
2132 }else{
2133     fmt.Printf("--I-- NG, could not joined (%v) %v\n",rcode,stat)
2134 }
2135 }
2136 func (gsh*GshContext)Rexec(argv[]string){
2137     if len(argv) <= 1 {
2138         fmt.Printf("--I-- rexec command [ | {file || {command} ]\n",gsh.RSERV)
2139         return
2140     }
2141 }
2142 /*
2143 nargv := gshScanArg(strings.Join(argv," "),0)
2144 fmt.Printf("--D-- nargc=%d [%v]\n",len(nargv),nargv)
2145 if nargv[1][0] != '{' {
2146     nargv[1] = "{" + nargv[1] + "}"
2147     fmt.Printf("--D-- nargc=%d [%v]\n",len(nargv),nargv)
2148 }
2149 argv = nargv
2150 */
2151 nargv := []string{}
2152 nargv = append(nargv,{"+"strings.Join(argv[1:], " ")+""})
2153 fmt.Printf("--D-- nargc=%d [%v]\n",len(nargv),nargv)
2154 argv = nargv
2155 }
2156 xargv := []string{"rex-exec","@"+gsh.RSERV,"GET"}
2157 xargv = append(xargv,argv...)
2158 xargv = append(xargv,"dev/tty")
2159 rcode,stat := gsh.RexecClient(xargv)
2160 if (rcode / 100) == 2 {
2161     fmt.Printf("--I-- OK Rexec (%v) %v\n",rcode,stat)
2162 }else{
2163     fmt.Printf("--I-- NG Rexec (%v) %v\n",rcode,stat)
2164 }
2165 }
2166 func (gsh*GshContext)Rchdir(argv[]string){
2167     if len(argv) <= 1 {
2168         return
2169     }
2170     cwd, _ := os.Getwd()
2171     os.Chdir(gsh.RWD)
2172     os.Chdir(argv[1])
2173     twd, _ := os.Getwd()
2174     gsh.RWD = twd
2175     fmt.Printf("--I-- JWD=%v\n",twd)
2176     os.Chdir(cwd)
2177 }
2178 func (gsh*GshContext)Rpwd(argv[]string){
2179     fmt.Printf("%v\n",gsh.RWD)
2180 }
2181 func (gsh*GshContext)Rls(argv[]string){
2182     cwd, _ := os.Getwd()
2183     os.Chdir(gsh.RWD)
2184     argv[0] = "-ls"
2185     gsh.xFind(argv)
2186     os.Chdir(cwd)
2187 }
2188 func (gsh*GshContext)Rput(argv[]string){
2189     var local string = ""
2190     var remote string = ""
2191     if 1 < len(argv) {
2192         local = argv[1]
2193         remote = local // base name
2194     }
2195     if 2 < len(argv) {
2196         remote = argv[2]
2197     }
2198     fmt.Printf("--I-- jput from=%v to=%v\n",local,gsh.Trelpath(remote))
2199 }
2200 func (gsh*GshContext)Rget(argv[]string){
2201     var remote string = ""
2202     var local string = ""
2203     if 1 < len(argv) {
2204         remote = argv[1]
2205         local = remote // base name
2206     }
2207     if 2 < len(argv) {
2208         local = argv[2]
2209     }
2210     fmt.Printf("--I-- jget from=%v to=%v\n",gsh.Trelpath(remote),local)
2211 }
2212 }
2213 // <a name="network">network</a>
2214 // -s, -si, -so // bi-directional, source, sync (maybe socket)
2215 func (gshCtx*GshContext)sconnect(inTCP bool, argv []string) {
2216     gshPA := gshCtx.gshPA
2217     if len(argv) < 2 {
2218         fmt.Printf("Usage: -s [host]:[port[.udp]]\n")
2219         return
2220     }
2221     remote := argv[1]
2222     if remote == ":" { remote = "0.0.0.0:9999" }
2223 }
2224 if inTCP { // TCP
2225     dport, err := net.ResolveTCPAddr("tcp",remote);
2226     if err != nil {
2227         fmt.Printf("Address error: %s (%s)\n",remote,err)
2228         return
2229     }
2230     conn, err := net.DialTCP("tcp",nil,dport)
2231     if err != nil {
2232         fmt.Printf("Connection error: %s (%s)\n",remote,err)
2233         return
2234     }
2235     file, _ := conn.File();
2236     fd := file.Fd()
2237     fmt.Printf("Socket: connected to %s, socket[%d]\n",remote,fd)
2238 }
2239 savfd := gshPA.Files[1]
2240 gshPA.Files[1] = fd;
2241 gshCtx.gshelly(argv[2:])
2242 gshPA.Files[1] = savfd
2243 file.Close()
2244 conn.Close()
2245 }else{
2246     //dport, err := net.ResolveUDPAddr("udp4",remote);
2247     dport, err := net.ResolveUDPAddr("udp",remote);
2248     if err != nil {
2249         fmt.Printf("Address error: %s (%s)\n",remote,err)

```

```

2250     return
2251 }
2252 //conn, err := net.DialUDP("udp4",nil,dport)
2253 conn, err := net.DialUDP("udp",nil,dport)
2254 if err != nil {
2255     fmt.Printf("Connection error: %s (%s)\n",remote,err)
2256     return
2257 }
2258 file, _ := conn.File();
2259 fd := file.Fd()
2260
2261 ar := conn.RemoteAddr()
2262 //al := conn.LocalAddr()
2263 fmt.Printf("Socket, connected to %s [%s], socket[%d]\n",
2264     remote,ar.String(),fd)
2265
2266 savfd := gshPA.Files[1]
2267 gshPA.Files[1] = fd;
2268 gshCtx.gshelly(argv[2:])
2269 gshPA.Files[1] = savfd
2270 file.Close()
2271 conn.Close()
2272 }
2273 }
2274 func (gshCtx*GshContext)saccept(inTCP bool, argv []string) {
2275     gshPA := gshCtx.gshPA
2276     if len(argv) < 2 {
2277         fmt.Printf("Usage: -ac [host]:[port].udp]\n")
2278         return
2279     }
2280     local := argv[1]
2281     if local == "" { local = "0.0.0.0:9999" }
2282     if inTCP { // TCP
2283         port, err := net.ResolveTCPAddr("tcp",local);
2284         if err != nil {
2285             fmt.Printf("Address error: %s (%s)\n",local,err)
2286             return
2287         }
2288         //fmt.Printf("Listen at %s...\n",local);
2289         sconn, err := net.ListenTCP("tcp", port)
2290         if err != nil {
2291             fmt.Printf("Listen error: %s (%s)\n",local,err)
2292             return
2293         }
2294         //fmt.Printf("Accepting at %s...\n",local);
2295         aconn, err := sconn.AcceptTCP()
2296         if err != nil {
2297             fmt.Printf("Accept error: %s (%s)\n",local,err)
2298             return
2299         }
2300         file, _ := aconn.File()
2301         fd := file.Fd()
2302         fmt.Printf("Accepted TCP at %s [%d]\n",local,fd)
2303
2304         savfd := gshPA.Files[0]
2305         gshPA.Files[0] = fd;
2306         gshCtx.gshelly(argv[2:])
2307         gshPA.Files[0] = savfd
2308
2309         sconn.Close();
2310         aconn.Close();
2311         file.Close();
2312     }else{
2313         //port, err := net.ResolveUDPAddr("udp4",local);
2314         port, err := net.ResolveUDPAddr("udp",local);
2315         if err != nil {
2316             fmt.Printf("Address error: %s (%s)\n",local,err)
2317             return
2318         }
2319         fmt.Printf("Listen UDP at %s...\n",local);
2320         //uconn, err := net.ListenUDP("udp4", port)
2321         uconn, err := net.ListenUDP("udp", port)
2322         if err != nil {
2323             fmt.Printf("Listen error: %s (%s)\n",local,err)
2324             return
2325         }
2326         file, _ := uconn.File()
2327         fd := file.Fd()
2328         ar := uconn.RemoteAddr()
2329         remote := ""
2330         if ar != nil { remote = ar.String() }
2331         if remote == "" { remote = "?" }
2332
2333         // not yet received
2334         //fmt.Printf("Accepted at %s [%d] <- %s\n",local,fd,"")
2335
2336         savfd := gshPA.Files[0]
2337         gshPA.Files[0] = fd;
2338         savenv := gshPA.Env
2339         gshPA.Env = append(savenv, "REMOTE_HOST="+remote)
2340         gshCtx.gshelly(argv[2:])
2341         gshPA.Env = savenv
2342         gshPA.Files[0] = savfd
2343
2344         uconn.Close();
2345         file.Close();
2346     }
2347 }
2348
2349 // empty line command
2350 func (gshCtx*GshContext)xPwd(argv[]string){
2351     // execute context command, pwd + date
2352     // context notation, representation scheme, to be resumed at re-login
2353     cwd, _ := os.Getwd()
2354     switch {
2355     case isin("-a",argv):
2356         gshCtx.ShowChdirHistory(argv)
2357     case isin("-ls",argv):
2358         showFileInfo(cwd,argv)
2359     default:
2360         fmt.Printf("%s\n",cwd)
2361     case isin("-v",argv): // obsolete empty command
2362         t := time.Now()
2363         date := t.Format(time.UnixDate)
2364         exe, _ := os.Executable()
2365         host, _ := os.Hostname()
2366         fmt.Printf("PWD=\"%s\"",cwd)
2367         fmt.Printf(" HOST=\"%s\"",host)
2368         fmt.Printf(" DATE=\"%s\"",date)
2369         fmt.Printf(" TIME=\"%s\"",t.String())
2370         fmt.Printf(" PID=\"%d\"",os.Getpid())
2371         fmt.Printf(" EXE=\"%s\"",exe)
2372         fmt.Printf("}\n")
2373     }
2374 }

```

```

2375
2376 // <a name="history">History</a>
2377 // these should be browsed and edited by HTTP browser
2378 // show the time of command with -t and direcotry with -ls
2379 // openfile-history, sort by -a -m -c
2380 // sort by elapsed time by -t -s
2381 // search by "more" like interface
2382 // edit history
2383 // sort history, and wc or uniq
2384 // CPU and other resource consumptions
2385 // limit showing range (by time or so)
2386 // export / import history
2387 func (gshCtx *GshContext)xHistory(argv []string){
2388     atWorkDirX := -1
2389     if 1 < len(argv) && strBegins(argv[1],"e") {
2390         atWorkDirX,_ = strconv.Atoi(argv[1][1:])
2391     }
2392     //fmt.Printf("--D-- showHistory(%v)\n",argv)
2393     for i, v := range gshCtx.CommandHistory {
2394         // exclude commands not to be listed by default
2395         // internal commands may be suppressed by default
2396         if v.CmdLine == "" && !isin("-a",argv) {
2397             continue;
2398         }
2399         if 0 <= atWorkDirX {
2400             if v.WorkDirX != atWorkDirX {
2401                 continue
2402             }
2403         }
2404         if !isin("-n",argv){ // like "fc"
2405             fmt.Printf("%!%-2d ",i)
2406         }
2407         if isin("-v",argv){
2408             fmt.Println(v) // should be with it date
2409         }else{
2410             if isin("-l",argv) || isin("-l0",argv) {
2411                 elps := v.EndAt.Sub(v.StartAt);
2412                 start := v.StartAt.Format(time.Stamp)
2413                 fmt.Printf("%d ",v.WorkDirX)
2414                 fmt.Printf("[%v] %11v/t ",start,elps)
2415             }
2416             if isin("-l",argv) && !isin("-l0",argv){
2417                 fmt.Printf("%v",Rusagef("%t %u\t// %s",argv,v.Rusagev))
2418             }
2419             if isin("-at",argv) { // isin("-ls",argv){
2420                 dhi := v.WorkDirX // workdir history index
2421                 fmt.Printf("%d %s\t",dhi,v.WorkDir)
2422                 // show the FileInfo of the output command??
2423             }
2424             fmt.Printf("%s",v.CmdLine)
2425             fmt.Printf("\n")
2426         }
2427     }
2428 }
2429 // ln - history index
2430 func searchHistory(gshCtx GshContext, gline string) (string, bool, bool){
2431     if gline[0] == '|' {
2432         hix, err := strconv.Atoi(gline[1:])
2433         if err != nil {
2434             fmt.Printf("--E-- (%s : range)\n",hix)
2435             return "", false, true
2436         }
2437         if hix < 0 || len(gshCtx.CommandHistory) <= hix {
2438             fmt.Printf("--E-- (%d : out of range)\n",hix)
2439             return "", false, true
2440         }
2441         return gshCtx.CommandHistory[hix].CmdLine, false, false
2442     }
2443     // search
2444     //for i, v := range gshCtx.CommandHistory {
2445     //}
2446     return gline, false, false
2447 }
2448 func (gsh*GshContext)cmdStringInHistory(hix int)(cmd string, ok bool){
2449     if 0 <= hix && hix < len(gsh.CommandHistory) {
2450         return gsh.CommandHistory[hix].CmdLine,true
2451     }
2452     return "",false
2453 }
2454
2455 // temporary adding to PATH environment
2456 // cd name -lib for LD LIBRARY_PATH
2457 // chdir with directory history (date + full-path)
2458 // -s for sort option (by visit date or so)
2459 func (gsh*GshContext)ShowChdirHistory(i int,v GChdirHistory, argv []string){
2460     fmt.Printf("%!%-2d ",v.CmdIndex) // the first command at this WorkDir
2461     fmt.Printf("%d ",i)
2462     fmt.Printf("[%v] ",v.MovedAt.Format(time.Stamp))
2463     showFileInfo(v.Dir,argv)
2464 }
2465 func (gsh*GshContext)ShowChdirHistory(argv []string){
2466     for i, v := range gsh.ChdirHistory {
2467         gsh.ShowChdirHistory(i,v,argv)
2468     }
2469 }
2470 func skipOpts(argv[]string)(int){
2471     for i,v := range argv {
2472         if strBegins(v,"-") {
2473             }else{
2474                 return i
2475             }
2476     }
2477     return -1
2478 }
2479 func (gshCtx*GshContext)xChdir(argv []string){
2480     cdhist := gshCtx.ChdirHistory
2481     if isin("?",argv) || isin("-t",argv) || isin("-a",argv) {
2482         gshCtx.ShowChdirHistory(argv)
2483         return
2484     }
2485     pwd, _ := os.Getwd()
2486     dir := ""
2487     if len(argv) <= 1 {
2488         dir = toFullpath("-")
2489     }else{
2490         i := skipOpts(argv[1:])
2491         if i < 0 {
2492             dir = toFullpath("-")
2493         }else{
2494             dir = argv[1+i]
2495         }
2496     }
2497     if strBegins(dir,"e") {
2498         if dir == "e0" { // obsolete
2499             dir = gshCtx.StartDir

```

```

2500     }else
2501     if dir == "@" {
2502         index := len(cdhist) - 1
2503         if 0 < index { index -= 1 }
2504         dir = cdhist[index].Dir
2505     }else{
2506         index, err := strconv.Atoi(dir[1:])
2507         if err != nil {
2508             fmt.Printf("--E-- xChdir(%v)\n",err)
2509             dir = "?"
2510         }else
2511         if len(gshCtx.ChdirHistory) <= index {
2512             fmt.Printf("--E-- xChdir(history range error)\n")
2513             dir = "?"
2514         }else{
2515             dir = cdhist[index].Dir
2516         }
2517     }
2518 }
2519 if dir != "?" {
2520     err := os.Chdir(dir)
2521     if err != nil {
2522         fmt.Printf("--E-- xChdir(%s)(%v)\n",argv[1],err)
2523     }else{
2524         cwd, _ := os.Getwd()
2525         if cwd != pwd {
2526             hist1 := GChdirHistory { }
2527             hist1.Dir = cwd
2528             hist1.MovedAt = time.Now()
2529             hist1.CmdIndex = len(gshCtx.CommandHistory)+1
2530             gshCtx.ChdirHistory = append(cdhist,hist1)
2531             if !isin("-s",argv){
2532                 //cwd, _ := os.Getwd()
2533                 //fmt.Printf("%s\n",cwd)
2534                 ix := len(gshCtx.ChdirHistory)-1
2535                 gshCtx.ShowChdirHistory1(ix,hist1,argv)
2536             }
2537         }
2538     }
2539 }
2540 if isin("-ls",argv){
2541     cwd, _ := os.Getwd()
2542     showFileInfo(cwd,argv);
2543 }
2544 }
2545 func TimeValSub(tv1 *syscall.Timeval, tv2 *syscall.Timeval){
2546     *tv1 = syscall.NsecToTimeval(tv1.Nano() - tv2.Nano())
2547 }
2548 func RusageSubv(ru1, ru2 [2]syscall.Rusage){[2]syscall.Rusage){
2549     TimeValSub(&ru1[0].Utime,&ru2[0].Utime)
2550     TimeValSub(&ru1[0].Stime,&ru2[0].Stime)
2551     TimeValSub(&ru1[1].Utime,&ru2[1].Utime)
2552     TimeValSub(&ru1[1].Stime,&ru2[1].Stime)
2553     return ru1
2554 }
2555 func TimeValAdd(tv1 syscall.Timeval, tv2 syscall.Timeval)(syscall.Timeval){
2556     tvs := syscall.NsecToTimeval(tv1.Nano() + tv2.Nano())
2557     return tvs
2558 }
2559 /*
2560 func RusageAddv(ru1, ru2 [2]syscall.Rusage){[2]syscall.Rusage){
2561     TimeValAdd(&ru1[0].Utime,&ru2[0].Utime)
2562     TimeValAdd(&ru1[0].Stime,&ru2[0].Stime)
2563     TimeValAdd(&ru1[1].Utime,&ru2[1].Utime)
2564     TimeValAdd(&ru1[1].Stime,&ru2[1].Stime)
2565     return ru1
2566 }
2567 */
2568
2569 // <a name="rusage">Resource Usage</a>
2570 func sRusagef(fmtspec string, argv []string, ru [2]syscall.Rusage)(string){
2571     // ru[0] self , ru[1] children
2572     ut := TimeValAdd(ru[0].Utime,ru[1].Utime)
2573     st := TimeValAdd(ru[0].Stime,ru[1].Stime)
2574     uu := (ut.Sec*1000000 + int64(ut.Usec)) * 1000
2575     su := (st.Sec*1000000 + int64(st.Usec)) * 1000
2576     tu := uu + su
2577     ret := fmt.Sprintf("%v/sum",abftime(tu))
2578     ret += fmt.Sprintf(" %v/usr",abftime(uu))
2579     ret += fmt.Sprintf(" %v/sys",abftime(su))
2580     return ret
2581 }
2582 func Rusagef(fmtspec string, argv []string, ru [2]syscall.Rusage)(string){
2583     ut := TimeValAdd(ru[0].Utime,ru[1].Utime)
2584     st := TimeValAdd(ru[0].Stime,ru[1].Stime)
2585     fmt.Printf("%d.%06ds/u ",ut.Sec,ut.Usec) //ru[1].Utime.Sec,ru[1].Utime.Usec)
2586     fmt.Printf("%d.%06ds/s ",st.Sec,st.Usec) //ru[1].Stime.Sec,ru[1].Stime.Usec)
2587     return ""
2588 }
2589 func Getrusagev(){[2]syscall.Rusage){
2590     var ruv = [2]syscall.Rusage{}
2591     syscall.Getrusage(syscall.RUSAGE_SELF,&ruv[0])
2592     syscall.Getrusage(syscall.RUSAGE_CHILDREN,&ruv[1])
2593     return ruv
2594 }
2595 func showRusage(what string,argv []string, ru *syscall.Rusage){
2596     fmt.Printf("%s: ",what);
2597     fmt.Printf("Utr=%d.%06ds",ru.Utime.Sec,ru.Utime.Usec)
2598     fmt.Printf(" Sys=%d.%06ds",ru.Stime.Sec,ru.Stime.Usec)
2599     fmt.Printf(" Rss=%vB",ru.Maxrss)
2600     if isin("-l",argv) {
2601         fmt.Printf(" MinFlt=%v",ru.Minflt)
2602         fmt.Printf(" MajFlt=%v",ru.Majflt)
2603         fmt.Printf(" IxRSS=%vB",ru.Ixrss)
2604         fmt.Printf(" IdRSS=%vB",ru.Idrss)
2605         fmt.Printf(" Nswap=%vB",ru.Nswap)
2606     }
2607     fmt.Printf(" Read=%v",ru.Inblock)
2608     fmt.Printf(" Write=%v",ru.Oublock)
2609 }
2610     fmt.Printf(" Snd=%v",ru.Msgsnd)
2611     fmt.Printf(" Rcv=%v",ru.Msgrcv)
2612     //if isin("-l",argv) {
2613         fmt.Printf(" Sig=%v",ru.Nsignals)
2614     }
2615 }
2616 func (gshCtx *GshContext)xTime(argv[]string)(bool){
2617     if 2 <= len(argv){
2618         gshCtx.LastRusage = syscall.Rusage{}
2619         rusagev1 := Getrusagev()
2620         fin := gshCtx.gshellv(argv[1:])
2621         rusagev2 := Getrusagev()
2622         showRusage(argv[1],argv,&gshCtx.LastRusage)
2623         rusagev := RusageSubv(rusagev2,rusagev1)
2624         showRusage("self",argv,&rusagev[0])

```

```

2625     showRusage("chld",argv,&rusagev[1])
2626     return fin
2627 }else{
2628     rusage:= syscall.Rusage {}
2629     syscall.Getrusage(syscall.RUSAGE_SELF,&rusage)
2630     showRusage("self",argv, &rusage)
2631     syscall.Getrusage(syscall.RUSAGE_CHILDREN,&rusage)
2632     showRusage("chld",argv, &rusage)
2633     return false
2634 }
2635 }
2636 func (gshCtx *GshContext)xJobs(argv[]string){
2637     fmt.Printf("%d Jobs\n",len(gshCtx.BackgroundJobs))
2638     for ji, pid := range gshCtx.BackgroundJobs {
2639         //wstat := syscall.WaitStatus (0)
2640         rusage := syscall.Rusage {}
2641         //wpid, err := syscall.Wait4(pid,&wstat,syscall.WNOHANG,&rusage);
2642         wpid, err := syscall.Wait4(pid,nil,syscall.WNOHANG,&rusage);
2643         if err != nil {
2644             fmt.Printf("--E-- %%%d [%d] (%v)\n",ji,pid,err)
2645         }else{
2646             fmt.Printf("%%d[%d](%d)\n",ji,pid,wpid)
2647             showRusage("chld",argv,&rusage)
2648         }
2649     }
2650 }
2651 func (gsh*GshContext)inBackground(argv[]string)(bool){
2652     if gsh.CmdTrace { fmt.Printf("--I-- inBackground(%v)\n",argv) }
2653     gsh.BackGround = true // set background option
2654     xfin := false
2655     xfin = gsh.gshellv(argv)
2656     gsh.BackGround = false
2657     return xfin
2658 }
2659 // -o file without command means just opening it and refer by #N
2660 // should be listed by "files" command
2661 func (gshCtx*GshContext)xOpen(argv[]string){
2662     var pv = []int{-1,-1}
2663     err := syscall.Pipe(pv)
2664     fmt.Printf("--I-- pipe()=[#%d,#%d](%v)\n",pv[0],pv[1],err)
2665 }
2666 func (gshCtx*GshContext)fromPipe(argv[]string){
2667 }
2668 func (gshCtx*GshContext)xClose(argv[]string){
2669 }
2670
2671 // <a name="redirect">redirect</a>
2672 func (gshCtx*GshContext)redirect(argv[]string)(bool){
2673     if len(argv) < 2 {
2674         return false
2675     }
2676
2677     cmd := argv[0]
2678     fname := argv[1]
2679     var file *os.File = nil
2680
2681     fdix := 0
2682     mode := os.O_RDONLY
2683
2684     switch {
2685     case cmd == "-i" || cmd == "<":
2686         fdix = 0
2687         mode = os.O_RDONLY
2688     case cmd == "-o" || cmd == ">":
2689         fdix = 1
2690         mode = os.O_RDWR | os.O_CREATE
2691     case cmd == "-a" || cmd == ">>":
2692         fdix = 1
2693         mode = os.O_RDWR | os.O_CREATE | os.O_APPEND
2694     }
2695     if fname[0] == '#' {
2696         fd, err := strconv.Atoi(fname[1:])
2697         if err != nil {
2698             fmt.Printf("--E-- (%v)\n",err)
2699             return false
2700         }
2701         file = os.NewFile(uintptr(fd),"MaybePipe")
2702     }else{
2703         xfile, err := os.OpenFile(argv[1], mode, 0600)
2704         if err != nil {
2705             fmt.Printf("--E-- (%s)\n",err)
2706             return false
2707         }
2708         file = xfile
2709     }
2710     gshPA := gshCtx.gshPA
2711     savfd := gshPA.Files[fdix]
2712     gshPA.Files[fdix] = file.Fd()
2713     fmt.Printf("--I-- Opened [%d] %s\n",file.Fd(),argv[1])
2714     gshCtx.gshellv(argv[2:])
2715     gshPA.Files[fdix] = savfd
2716
2717     return false
2718 }
2719
2720 //fmt.Fprintf(res, "GShell Status: %q", html.EscapeString(req.URL.Path))
2721 func httpHandler(res http.ResponseWriter, req *http.Request){
2722     path := req.URL.Path
2723     fmt.Printf("--I-- Got HTTP Request(%s)\n",path)
2724     {
2725         gshCtxBuf, _ := setupGshContext()
2726         gshCtx := &gshCtxBuf
2727         fmt.Printf("--I-- %s\n",path[1:])
2728         gshCtx.tgshelll(path[1:])
2729     }
2730     fmt.Fprintf(res, "Hello(^-^)/\n%s\n",path)
2731 }
2732 func (gshCtx *GshContext) httpServer(argv []string){
2733     http.HandleFunc("/", httpHandler)
2734     accport := "localhost:9999"
2735     fmt.Printf("--I-- HTTP Server Start at [%s]\n",accport)
2736     http.ListenAndServe(accport,nil)
2737 }
2738 func (gshCtx *GshContext)xGo(argv[]string){
2739     go gshCtx.gshellv(argv[1:]);
2740 }
2741 func (gshCtx *GshContext) xPs(argv[]string)(){
2742 }
2743
2744 // <a name="plugin">Plugin</a>
2745 // plugin [-ls [names]] to list plugins
2746 // Reference: <a href="https://golang.org/src/plugin/">plugin</a> source code
2747 func (gshCtx *GshContext) whichPlugin(name string,argv[]string)(pi *PluginInfo){
2748     pi = nil
2749     for _,p := range gshCtx.PluginFuncs {

```

```

2750     if p.Name == name && pi == nil {
2751         pi = *p
2752     }
2753     if !isin("-s",argv){
2754         //fmt.Printf("%v %v ",i,p)
2755         if isin("-ls",argv){
2756             showFileInfo(p.Path,argv)
2757         }else{
2758             fmt.Printf("%s\n",p.Name)
2759         }
2760     }
2761 }
2762 return pi
2763 }
2764 func (gshCtx *GshContext) xPlugin(argv[]string) (error) {
2765     if len(argv) == 0 || argv[0] == "-ls" {
2766         gshCtx.whichPlugin("",argv)
2767         return nil
2768     }
2769     name := argv[0]
2770     pin := gshCtx.whichPlugin(name,[]string{"-s"})
2771     if pin != nil {
2772         os.Args = argv // should be recovered?
2773         pin.Addr.(func())()
2774         return nil
2775     }
2776     sofile := toFullpath(argv[0] + ".so") // or find it by which($PATH)
2777
2778     p, err := plugin.Open(sofile)
2779     if err != nil {
2780         fmt.Printf("--E-- plugin.Open(%s)(%v)\n",sofile,err)
2781         return err
2782     }
2783     fname := "Main"
2784     f, err := p.Lookup(fname)
2785     if( err != nil ){
2786         fmt.Printf("--E-- plugin.Lookup(%s)(%v)\n",fname,err)
2787         return err
2788     }
2789     pin := PluginInfo {p,f,name,sofile}
2790     gshCtx.PluginFuncs = append(gshCtx.PluginFuncs,pin)
2791     fmt.Printf("--I-- added (%d)\n",len(gshCtx.PluginFuncs))
2792
2793     //fmt.Printf("--I-- first call(%s:%s)%v\n",sofile,fname,argv)
2794     os.Args = argv
2795     f.(func())()
2796     return err
2797 }
2798 func (gshCtx*GshContext)Args(argv[]string){
2799     for i,v := range os.Args {
2800         fmt.Printf("[%v] %v\n",i,v)
2801     }
2802 }
2803 func (gshCtx *GshContext) showVersion(argv[]string){
2804     if isin("-l",argv) {
2805         fmt.Printf("%v/%v (%v)",NAME,VERSION,DATE);
2806     }else{
2807         fmt.Printf("%v",VERSION);
2808     }
2809     if isin("-a",argv) {
2810         fmt.Printf(" %s",AUTHOR)
2811     }
2812     if !isin("-n",argv) {
2813         fmt.Printf("\n")
2814     }
2815 }
2816
2817 // <a name="scanf">Scanf</a> // string decomposer
2818 // scanf [format] [input]
2819 func scanv(sstr string)(strv[]string){
2820     strv = strings.Split(sstr, " ")
2821     return strv
2822 }
2823 func scanUntil(src,end string)(rstr string,leng int){
2824     idx := strings.Index(src,end)
2825     if 0 <= idx {
2826         rstr = src[0:idx]
2827         return rstr,idx+leng(end)
2828     }
2829     return src,0
2830 }
2831
2832 // -bn -- display base-name part only // can be in some %fmt, for sed rewriting
2833 func (gsh*GshContext)printVal(fmts string, vstr string, optv[]string){
2834     //vint,err := strconv.Atoi(vstr)
2835     var ival int64 = 0
2836     n := 0
2837     err := error(nil)
2838     if strBegins(vstr, "_") {
2839         vx, _ := strconv.Atoi(vstr[1:])
2840         if vx < len(gsh.iValues) {
2841             vstr = gsh.iValues[vx]
2842         }else{
2843         }
2844     }
2845     // should use Eval()
2846     if strBegins(vstr,"0x") {
2847         n,err = fmt.Sscanf(vstr[2:],"%x",&ival)
2848     }else{
2849         n,err = fmt.Sscanf(vstr,"%d",&ival)
2850     }//fmt.Printf("--D-- n=%d err=(%v) (%s)=%v\n",n,err,vstr, ival)
2851 }
2852 if n == 1 && err == nil {
2853     //fmt.Printf("--D-- formatn(%v) ival(%v)\n",fmts,ival)
2854     fmt.Printf("%"+fmts,ival)
2855 }else{
2856     if isin("-bn",optv){
2857         fmt.Printf("%"+fmts,filepath.Base(vstr))
2858     }else{
2859         fmt.Printf("%"+fmts,vstr)
2860     }
2861 }
2862 }
2863 func (gsh*GshContext)printfv(fmts,div string,argv[]string,optv[]string,list[]string){
2864     //fmt.Printf("%d",len(list))
2865     //curfmt := "%v"
2866     outlen := 0
2867     curfmt := gsh.iFormat
2868
2869     if 0 < len(fmts) {
2870         for xi := 0; xi < len(fmts); xi++ {
2871             fch := fmts[xi]
2872             if fch == '%' {
2873                 if xi+1 < len(fmts) {
2874                     curfmt = string(fmts[xi+1])

```

```

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

```



```

3000     gsh.iValStack = append(gsh.iValStack,argv[1:])
3001     fmt.Printf("depth=%d\n",len(gsh.iValStack))
3002 }
3003 func (gsh*GshContext)Dump(argv[]string){
3004     for i,v := range gsh.iValStack {
3005         fmt.Printf("%d %v\n",i,v)
3006     }
3007 }
3008 func (gsh*GshContext)Pop(argv[]string){
3009     depth := len(gsh.iValStack)
3010     if 0 < depth {
3011         v := gsh.iValStack[depth-1]
3012         if !isn("-cat",argv){
3013             gsh.iValues = append(gsh.iValues,v...)
3014         }else{
3015             gsh.iValues = v
3016         }
3017         gsh.iValStack = gsh.iValStack[0:depth-1]
3018         fmt.Printf("depth=%d %s\n",len(gsh.iValStack),gsh.iValues)
3019     }else{
3020         fmt.Printf("depth=%d\n",depth)
3021     }
3022 }
3023 }
3024 // <a name="interpreter">Command Interpreter</a>
3025 func (gshCtx*GshContext)gshellv(argv []string) (fin bool) {
3026     fin = false
3027
3028     if gshCtx.CmdTrace { fmt.Fprintf(os.Stderr,"--I-- gshellv(%d)\n",len(argv)) }
3029     if len(argv) <= 0 {
3030         return false
3031     }
3032     xargv := []string{}
3033     for ai := 0; ai < len(argv); ai++ {
3034         xargv = append(xargv,subst(gshCtx,argv[ai],false))
3035     }
3036     argv = xargv
3037     if false {
3038         for ai := 0; ai < len(argv); ai++ {
3039             fmt.Printf("[%d] %s [%d]\n",
3040                 ai,argv[ai],len(argv[ai]),argv[ai])
3041         }
3042     }
3043     cmd := argv[0]
3044     if gshCtx.CmdTrace { fmt.Fprintf(os.Stderr,"--I-- gshellv(%d)\n",len(argv),argv) }
3045     switch { // https://tour.golang.org/flowcontrol/11
3046     case cmd == "":
3047         gshCtx.xPwd([]string{}); // empty command
3048     case cmd == "-x":
3049         gshCtx.CmdTrace = ! gshCtx.CmdTrace
3050     case cmd == "-xt":
3051         gshCtx.CmdTime = ! gshCtx.CmdTime
3052     case cmd == "-ot":
3053         gshCtx.sconnect(true, argv)
3054     case cmd == "-ou":
3055         gshCtx.sconnect(false, argv)
3056     case cmd == "-it":
3057         gshCtx.saccept(true, argv)
3058     case cmd == "-iu":
3059         gshCtx.saccept(false, argv)
3060     case cmd == "-i" || cmd == "<" || cmd == "-o" || cmd == ">" || cmd == "-a" || cmd == ">>" || cmd == "-s" || cmd == "><":
3061         gshCtx.redirect(argv)
3062     case cmd == "|":
3063         gshCtx.fromPipe(argv)
3064     case cmd == "args":
3065         gshCtx.Args(argv)
3066     case cmd == "bg" || cmd == "-bg":
3067         rfin := gshCtx.inBackground(argv[1:])
3068         return rfin
3069     case cmd == "-bn":
3070         gshCtx.BaseName(argv)
3071     case cmd == "call":
3072         _ = gshCtx.excommand(false,argv[1:])
3073     case cmd == "cd" || cmd == "chdir":
3074         gshCtx.xChdir(argv);
3075     case cmd == "-cksum":
3076         gshCtx.xFind(argv)
3077     case cmd == "-sum":
3078         gshCtx.xFind(argv)
3079     case cmd == "close":
3080         gshCtx.xClose(argv)
3081     case cmd == "gcp":
3082         gshCtx.FileCopy(argv)
3083     case cmd == "dec" || cmd == "decode":
3084         gshCtx.Dec(argv)
3085     case cmd == "#define":
3086     case cmd == "dump":
3087         gshCtx.Dump(argv)
3088     case cmd == "echo":
3089         echo(argv,true)
3090     case cmd == "enc" || cmd == "encode":
3091         gshCtx.Enc(argv)
3092     case cmd == "env":
3093         env(argv)
3094     case cmd == "eval":
3095         xEval(argv[1:],true)
3096     case cmd == "exec":
3097         _ = gshCtx.excommand(true,argv[1:])
3098         // should not return here
3099     case cmd == "exit" || cmd == "quit":
3100         // write Result code EXIT to >
3101         return true
3102     case cmd == "fds":
3103         // dump the attributes of fds (of other process)
3104     case cmd == "-find" || cmd == "fin" || cmd == "ufind" || cmd == "uf":
3105         gshCtx.xFind(argv[1:])
3106     case cmd == "fu":
3107         gshCtx.xFind(argv[1:])
3108     case cmd == "fork":
3109         // mainly for a server
3110     case cmd == "-gen":
3111         gshCtx.gen(argv)
3112     case cmd == "-go":
3113         gshCtx.xGo(argv)
3114     case cmd == "-grep":
3115         gshCtx.xFind(argv)
3116     case cmd == "gdeg":
3117         gshCtx.Deg(argv)
3118     case cmd == "genq":
3119         gshCtx.Eng(argv)
3120     case cmd == "gpop":
3121         gshCtx.Pop(argv)
3122     case cmd == "gpush":
3123         gshCtx.Push(argv)
3124     case cmd == "history" || cmd == "hi": // hi should be alias

```

```

3125     gshCtx.xHistory(argv)
3126 case cmd == "jobs":
3127     gshCtx.xJobs(argv)
3128 case cmd == "insp":
3129     gshCtx.SplitLine(argv)
3130 case cmd == "-ls":
3131     gshCtx.xFind(argv)
3132 case cmd == "nop":
3133     // do nothing
3134 case cmd == "pipe":
3135     gshCtx.xOpen(argv)
3136 case cmd == "plug" || cmd == "plugin" || cmd == "pin":
3137     gshCtx.xPlug(argv[1:])
3138 case cmd == "print" || cmd == "-pr":
3139     // output internal slice // also sprintf should be
3140     gshCtx.Printv(argv)
3141 case cmd == "ps":
3142     gshCtx.xPs(argv)
3143 case cmd == "pstitle":
3144     // to be gsh.title
3145 case cmd == "rexeed" || cmd == "rexd":
3146     gshCtx.RexecServer(argv)
3147 case cmd == "rexec" || cmd == "rex":
3148     gshCtx.RexecClient(argv)
3149 case cmd == "repeat" || cmd == "rep": // repeat cond command
3150     gshCtx.repeat(argv)
3151 case cmd == "scan":
3152     // scan input (or so in fscanf) to internal slice (like Files or map)
3153     gshCtx.Scanv(argv)
3154 case cmd == "set":
3155     // set name ...
3156 case cmd == "serv":
3157     gshCtx.httpServer(argv)
3158 case cmd == "shift":
3159     gshCtx.Shiftv(argv)
3160 case cmd == "sleep":
3161     gshCtx.sleep(argv)
3162 case cmd == "-sort":
3163     gshCtx.Sortv(argv)
3164
3165 case cmd == "j" || cmd == "join":
3166     gshCtx.Rjoin(argv)
3167 case cmd == "a" || cmd == "alpa":
3168     gshCtx.Rexec(argv)
3169 case cmd == "jcd" || cmd == "jchdir":
3170     gshCtx.Rchdir(argv)
3171 case cmd == "jget":
3172     gshCtx.Rget(argv)
3173 case cmd == "jle":
3174     gshCtx.Rls(argv)
3175 case cmd == "jput":
3176     gshCtx.Rput(argv)
3177 case cmd == "jpwd":
3178     gshCtx.Rpwd(argv)
3179
3180 case cmd == "time":
3181     fin = gshCtx.xTime(argv)
3182 case cmd == "pwd":
3183     gshCtx.xPwd(argv);
3184 case cmd == "ver" || cmd == "-ver" || cmd == "version":
3185     gshCtx.showVersion(argv)
3186 case cmd == "where":
3187     // data file or so?
3188 case cmd == "which":
3189     which("PATH",argv);
3190 default:
3191     if gshCtx.whichPlugin(cmd,[]string{"-s"}) != nil {
3192         gshCtx.xPlugin(argv)
3193     }else{
3194         notfound,_ := gshCtx.excommand(false,argv)
3195         if notfound {
3196             fmt.Printf("--E-- command not found (%v)\n",cmd)
3197         }
3198     }
3199 }
3200 return fin
3201 }
3202
3203 func (gsh*GshContext)gshell(gline string) (rfin bool) {
3204     argv := strings.Split(string(gline)," ")
3205     fin := gsh.gshellv(argv)
3206     return fin
3207 }
3208 func (gsh*GshContext)tgshell(gline string)(xfin bool){
3209     start := time.Now()
3210     fin := gsh.gshell(gline)
3211     end := time.Now()
3212     elps := end.Sub(start);
3213     if gsh.CmdTime {
3214         fmt.Printf("--T-- " + time.Now().Format(time.Stamp) + " (%d.%09ds)\n",
3215             elps/1000000000,elps%1000000000)
3216     }
3217     return fin
3218 }
3219 func Ttyid() (int) {
3220     fi, err := os.Stdin.Stat()
3221     if err != nil {
3222         return 0;
3223     }
3224     //fmt.Printf("Stdin: %v Dev=%d\n",
3225     // fi.Mode(),fi.Mode()&os.ModeDevice)
3226     if (fi.Mode() & os.ModeDevice) != 0 {
3227         stat := syscall.Stat_t{};
3228         err := syscall.Fstat(0,&stat)
3229         if err != nil {
3230             //fmt.Printf("--I-- Stdin: (%v)\n",err)
3231         }else{
3232             //fmt.Printf("--I-- Stdin: rdev=%d %d\n",
3233             // stat.Rdev&0xFF,stat.Rdev);
3234             //fmt.Printf("--I-- Stdin: tty=%d\n",stat.Rdev&0xFF);
3235             return int(stat.Rdev & 0xFF)
3236         }
3237     }
3238     return 0
3239 }
3240 func (gshCtx *GshContext) ttyfile() string {
3241     //fmt.Printf("--I-- GSH_HOME=%s\n",gshCtx.GshHomeDir)
3242     ttyfile := gshCtx.GshHomeDir + "/" + "gsh-tty" +
3243         fmt.Sprintf("%02d",gshCtx.TerminalId)
3244     //strconv.Itoa(gshCtx.TerminalId)
3245     //fmt.Printf("--I-- ttyfile=%s\n",ttyfile)
3246     return ttyfile
3247 }
3248 func (gshCtx *GshContext) ttyline>(*os.File){
3249     file, err := os.OpenFile(gshCtx.ttyfile(),os.O_RDWR|os.O_CREATE|os.O_TRUNC,0600)

```

```

3250 if err != nil {
3251     fmt.Printf("--F-- cannot open %s (%s)\n",gshCtx.ttyfile(),err)
3252     return file;
3253 }
3254 return file
3255 }
3256 func (gshCtx *GshContext)getline(hix int, skipping bool, prevline string) (string) {
3257     if( skipping ){
3258         reader := bufio.NewReaderSize(os.Stdin,LINESIZE)
3259         line, _, _ := reader.ReadLine()
3260         return string(line)
3261     }else{
3262         if true {
3263             return xgetline(hix,prevline,gshCtx)
3264         }
3265         /*
3266         else
3267         if( with_exgetline && gshCtx.GetLine != "" ){
3268             //var xhix int64 = int64(hix); // cast
3269             newenv := os.Environ()
3270             newenv = append(newenv, "GSH_LINENO="+strconv.FormatInt(int64(hix),10) )
3271
3272             tty := gshCtx.ttyline()
3273             tty.WriteString(prevline)
3274             Pa := os.ProcAttr {
3275                 "", // start dir
3276                 newenv, //os.Environ(),
3277                 []*os.File{os.Stdin,os.Stdout,os.Stderr,tty},
3278                 nil,
3279             }
3280             //fmt.Printf("--I-- getline=%s // %s\n",gsh_getlinev[0],gshCtx.GetLine)
3281             proc, err := os.StartProcess(gsh_getlinev[0],[]string{"getline","getline"},&Pa)
3282             if err != nil {
3283                 fmt.Printf("--F-- getline process error (%v)\n",err)
3284                 // for i; {
3285                 return "exit (getline program failed)"
3286             }
3287             //stat, err := proc.Wait()
3288             proc.Wait()
3289             buff := make([]byte,LINESIZE)
3290             count, err := tty.Read(buff)
3291             //_, err = tty.Read(buff)
3292             //fmt.Printf("--D-- getline (%d)\n",count)
3293             if err != nil {
3294                 if ! (count == 0) { // && err.String() == "EOF" } {
3295                     fmt.Printf("--E-- getline error (%s)\n",err)
3296                 }
3297             }else{
3298                 //fmt.Printf("--I-- getline OK \"%s\"\n",buff)
3299             }
3300             tty.Close()
3301             gline := string(buff[0:count])
3302             return gline
3303         }else
3304         */
3305         {
3306             // if isatty {
3307             fmt.Printf("%d",hix)
3308             fmt.Print(PROMPT)
3309             // }
3310             reader := bufio.NewReaderSize(os.Stdin,LINESIZE)
3311             line, _, _ := reader.ReadLine()
3312             return string(line)
3313         }
3314     }
3315 }
3316 //== begin ===== getline
3317 /*
3318 * getline.c
3319 * 2020-0819 extracted from dog.c
3320 * getline.go
3321 * 2020-0822 ported to Go
3322 */
3323 /*
3324 package main // getline main
3325 import (
3326     "fmt" // <a href="https://golang.org/pkg/fmt/">fmt</a>
3327     "strings" // <a href="https://golang.org/pkg/strings/">strings</a>
3328     "os" // <a href="https://golang.org/pkg/os/">os</a>
3329     "syscall" // <a href="https://golang.org/pkg/syscall/">syscall</a>
3330     //"bytes" // <a href="https://golang.org/pkg/os/">os</a>
3331     //"os/exec" // <a href="https://golang.org/pkg/os/">os</a>
3332 )
3333 */
3334
3335 // C language compatibility functions
3336 var errno = 0
3337 var stdin *os.File = os.Stdin
3338 var stdout *os.File = os.Stdout
3339 var stderr *os.File = os.Stderr
3340 var EOF = -1
3341 var NULL = 0
3342 type FILE os.File
3343 type StrBuff []byte
3344 var NULL_FP *os.File = nil
3345 var NULLSP = 0
3346 //var LINESIZE = 1024
3347
3348 func system(cmdstr string)(int){
3349     PA := syscall.ProcAttr {
3350         "", // the starting directory
3351         os.Environ(),
3352         []uintptr{os.Stdin.Fd(),os.Stdout.Fd(),os.Stderr.Fd()},
3353         nil,
3354     }
3355     argv := strings.Split(cmdstr, " ")
3356     pid,err := syscall.ForkExec(argv[0],argv,&PA)
3357     if( err != nil ){
3358         fmt.Printf("--E-- syscall(%v) err(%v)\n",cmdstr,err)
3359     }
3360     syscall.Wait4(pid,nil,0,nil)
3361
3362     /*
3363     argv := strings.Split(cmdstr, " ")
3364     fmt.Fprintf(os.Stderr,"--I-- system(%v)\n",argv)
3365     //cmd := exec.Command(argv[0]...)
3366     cmd := exec.Command(argv[0],argv[1],argv[2])
3367     cmd.Stdin = strings.NewReader("output of system")
3368     var out bytes.Buffer
3369     cmd.Stdout = &out
3370     var serr bytes.Buffer
3371     cmd.Stderr = &serr
3372     err := cmd.Run()
3373     if err != nil {
3374         fmt.Fprintf(os.Stderr,"--E-- system(%v)err(%v)\n",argv,err)

```

```

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

```

```

3500 func fputc(ch int , fp*os.File)(int){
3501     var buf [1]byte
3502     buf[0] = byte(ch)
3503     fp.Write(buf[0:1])
3504     return 0
3505 }
3506 func fputs(buf StrBuff, fp*os.File)(int){
3507     fp.Write(buf)
3508     return 0
3509 }
3510 func xfputss(str string, fp*os.File)(int){
3511     return fputs([]byte(str),fp)
3512 }
3513 func sscanf(str StrBuff,fmts string, params ...interface{})(int){
3514     fmt.Sscanf(string(str[0:strlen(str)]),fmts,params...)
3515     return 0
3516 }
3517 func fprintf(fp*os.File,fmts string, params ...interface{})(int){
3518     fmt.Fprintf(fp,fmts,params...)
3519     return 0
3520 }
3521
3522 // <a name="IME">Command Line IME</a>
3523 //----- MyIME
3524 var MyIMEVER = "MyIME/0.0.2";
3525 type RomKana struct {
3526     pat string;
3527     out string;
3528 }
3529 var dicents = 0
3530 var romkana [1024]RomKana
3531 func readDic()(int){
3532     var rk *os.File;
3533     var dic = "MyIME-dic.txt";
3534     //rk = fopen("romkana.txt","r");
3535     //rk = fopen("JK-JA-morse-dic.txt","r");
3536     rk = fopen(dic,"r");
3537     if( rk == NULL_FP ){
3538         if( true ){
3539             fprintf(stderr,"--%s-- Could not load %s\n",MyIMEVER,dic);
3540         }
3541         return -1;
3542     }
3543     if( true ){
3544         var di int;
3545         var line = make(StrBuff,1024);
3546         var pat string
3547         var out string;
3548         for di = 0; di < 1024; di++ {
3549             if( fgets(line,sizeof(line),rk) == NULLSP ){
3550                 break;
3551             }
3552             fmt.Sscanf(string(line[0:strlen(line)]),"%s %s",&pat,&out);
3553             //sscanf(line,"%s %[\r\n]",&pat,&out);
3554             romkana[di].pat = pat;
3555             romkana[di].out = out;
3556             //fprintf(stderr,"--Dd- %s %s\n",pat,out)
3557         }
3558         dicents += di
3559         if( false ){
3560             fprintf(stderr,"--%s-- loaded romkana.txt [%d]\n",MyIMEVER,di);
3561             for di = 0; di < dicents; di++ {
3562                 fprintf(stderr,
3563                     "%s %s\n",romkana[di].pat,romkana[di].out);
3564             }
3565         }
3566     }
3567     fclose(rk);
3568
3569     //romkana[dicents].pat = "//ddump"
3570     //romkana[dicents].pat = "//ddump" // dump the dic. and clean the command input
3571     return 0;
3572 }
3573 func matchlen(stri string, pati string)(int){
3574     if strBegins(stri,pati) {
3575         return len(pati)
3576     }else{
3577         return 0
3578     }
3579 }
3580 func convs(src string)(string){
3581     var si int;
3582     var sx = len(src);
3583     var di int;
3584     var mi int;
3585     var dstb []byte
3586
3587     for si = 0; si < sx; { // search max. match from the position
3588         if strBegins(src[si:], "%x/") {
3589             // %x/integer/ // s/a/b/
3590             ix := strings.Index(src[si+3:], "/")
3591             if 0 < ix {
3592                 var iv int = 0
3593                 //fmt.Sscanf(src[si+3:si+3+ix], "%d", &iv)
3594                 fmt.Sscanf(src[si+3:si+3+ix], "%v", &iv)
3595                 sval := fmt.Sprintf("%x", iv)
3596                 bval := []byte(sval)
3597                 dstb = append(dstb, bval...)
3598                 si = si+3+ix+1
3599                 continue
3600             }
3601         }
3602         if strBegins(src[si:], "%d/") {
3603             // %d/integer/ // s/a/b/
3604             ix := strings.Index(src[si+3:], "/")
3605             if 0 < ix {
3606                 var iv int = 0
3607                 fmt.Sscanf(src[si+3:si+3+ix], "%v", &iv)
3608                 sval := fmt.Sprintf("%d", iv)
3609                 bval := []byte(sval)
3610                 dstb = append(dstb, bval...)
3611                 si = si+3+ix+1
3612                 continue
3613             }
3614         }
3615         var maxlen int = 0;
3616         var len int;
3617         mi = -1;
3618         for di = 0; di < dicents; di++ {
3619             len = matchlen(src[si:], romkana[di].pat);
3620             if( maxlen < len ){
3621                 maxlen = len;
3622                 mi = di;
3623             }
3624         }

```

```

3625     if( 0 < maxlen ){
3626         out := romkana[mi].out;
3627         dstb = append(dstb,[]byte(out)...);
3628         si += maxlen;
3629     }else{
3630         dstb = append(dstb,src[si])
3631         si += 1;
3632     }
3633 }
3634 return string(dstb)
3635 }
3636 func trans(src string)(int){
3637     dst := convs(src);
3638     xputss(dst,stderr);
3639     return 0;
3640 }
3641
3642 //----- LINEEDIT
3643 // "?" at the top of the line means searching history
3644
3645 var GO_UP = 201
3646 var GO_DOWN = 202
3647 var GO_RIGHT = 203
3648 var GO_LEFT = 204
3649
3650 func getesc(in *os.File)(int){
3651     var ch1 int
3652     var ch2 int
3653     ch1 = fgetc(in);
3654     ch2 = fgetc(in);
3655     if false {
3656         fprintf(stderr,"%c/%X %c/%X",ch1,ch1,ch2,ch2);
3657     }
3658     switch( ch1 ){
3659     case '[':
3660         switch( ch2 ){
3661             case 'A': return GO_UP; // ^
3662             case 'B': return GO_DOWN; // v
3663             case 'C': return GO_RIGHT; // >
3664             case 'D': return GO_LEFT; // <
3665         }
3666         break;
3667     }
3668     return 0;
3669 }
3670 func clearline(){
3671     var i int
3672     fprintf(stderr,"\r");
3673     for i = 0; i < 80; i++ {
3674         fputc(' ',os.Stderr);
3675     }
3676     fprintf(stderr,"\r");
3677 }
3678 var romkanmode bool;
3679 var insertmode int;
3680 func redraw(lno int,line string,right string){
3681     var bsi int
3682     var rlen int
3683     var romkanmark string
3684
3685     if( romkanmode ){
3686         //romkanmark = " *";
3687     }else{
3688         romkanmark = "";
3689     }
3690     clearline();
3691     xputss("\r",stderr);
3692     if( romkanmode ){
3693         fprintf(stderr,"[\343\201\202r]");
3694         //fprintf(stderr,"[R]");
3695     }
3696     fprintf(stderr,"%d! ",lno);
3697     if( romkanmode ){
3698         trans(line);
3699         //fputs(romkanmark,stderr);
3700         trans(right);
3701     }else{
3702         xputss(line,stderr);
3703         //fputs(romkanmark,stderr);
3704         xputss(right,stderr);
3705     }
3706     if true { //romkanmode {
3707         fprintf(stderr,"\r")
3708         if romkanmode {
3709             fprintf(stderr,"[\343\201\202r]");
3710             fprintf(stderr,"%d! ",lno);
3711             trans(line);
3712         }else{
3713             fprintf(stderr,"%d! ",lno);
3714             xputss(line,stderr);
3715         }
3716     }else{
3717         rlen = len(right) + len(romkanmark);
3718         if true {
3719             for bsi = 0; bsi < rlen; bsi++ {
3720                 fputc('\b',stderr);
3721             }
3722         }
3723     }
3724 }
3725 func delHeadChar(str string)(rline string,head string){
3726     clen := utf8.DecodeRune([]byte(str))
3727     head = string(str[0:clen])
3728     return str[clen:],head
3729 }
3730 func delTailChar(str string)(rline string, last string){
3731     var i = 0
3732     var clen = 0
3733     for {
3734         ,siz := utf8.DecodeRune([]byte(str)[i:])
3735         if siz <= 0 { break }
3736         clen = siz
3737         i += siz
3738     }
3739     last = str[len(str)-clen:]
3740     return str[0:len(str)-clen],last
3741 }
3742
3743 // 3> for output and history
3744 // 4> for keylog?
3745 // <a name="getline">Command Line Editor</a>
3746 func xgetline(lno int, prevline string, gsh*GshContext)(string){
3747     lastlno := lno;
3748     line := ""
3749     right := ""

```

```

3750
3751 //readDic();
3752 if( isatty(0) == 0 ){
3753     if( sifgets(&line,LINESIZE,stdin) == NULL ){
3754         line = "exit\n";
3755     }else{
3756     }
3757     goto EXIT_GOT;
3758 }
3759 if( true ){
3760     //var pts string;
3761     //pts = ptsname(0);
3762     //pts = ttyname(0);
3763     //fprintf(stderr,"--pts[0] = %s\n,pts?pts:?"");
3764 }
3765 if( false ){
3766     fprintf(stderr,"! ");
3767     fflush(stderr);
3768     sfgets(&line,LINESIZE,stdin);
3769 }else{
3770     var ch int;
3771
3772     system("/bin/stty -echo -icanon");
3773     redraw(lno,line,right);
3774     line = ""
3775     right = ""
3776     pch := -1
3777     for {
3778         if( pch != -1 ){
3779             ch = pch
3780             pch = -1
3781         }else{
3782             ch = fgetc(stdin);
3783         }
3784         if( ch == 033 ){
3785             ch = getesc(stdin);
3786         }
3787         if( ch == '\\ '){
3788             fputc(ch,stderr)
3789             ch = fgetc(stdin)
3790             if( ch == 'j' || ch == 'J' ){
3791                 readDic();
3792                 romkanmode = !romkanmode;
3793                 if( ch == 'J' ){
3794                     fprintf(stderr,"J\r\n");
3795                 }
3796                 redraw(lno,line,right);
3797                 continue
3798             }else{
3799                 if( ch == 'i' || ch == 'I' ){
3800                     dst := convs(line+right);
3801                     line = dst
3802                     right = ""
3803                     if( ch == 'I' ){
3804                         fprintf(stderr,"I\r\n");
3805                     }
3806                     redraw(lno,line,right);
3807                     continue
3808                 }else{
3809                     pch = ch
3810                     ch = '\\ '
3811                 }
3812             }
3813             switch( ch ){
3814                 case 0:
3815                     continue;
3816                 case GO_UP:
3817                     if lno == 1 {
3818                         continue
3819                     }
3820                     cmd,ok := gsh.cmdStringInHistory(lno-1)
3821                     if ok {
3822                         line = cmd
3823                         right = ""
3824                         lno = lno - 1
3825                     }
3826                     redraw(lno,line,right);
3827                     continue
3828                 case GO_DOWN:
3829                     cmd,ok := gsh.cmdStringInHistory(lno+1)
3830                     if ok {
3831                         line = cmd
3832                         right = ""
3833                         lno = lno + 1
3834                     }else{
3835                         line = ""
3836                         right = ""
3837                         if lno == lastlno-1 {
3838                             lno = lno + 1
3839                         }
3840                     }
3841                     redraw(lno,line,right);
3842                     continue
3843                 case GO_LEFT:
3844                     if 0 < len(line) {
3845                         xline,tail := delTailChar(line)
3846                         line = xline
3847                         right = tail + right
3848                     }
3849                     redraw(lno,line,right);
3850                     continue;
3851                 case GO_RIGHT:
3852                     if( 0 < len(right) && right[0] != 0 ){
3853                         xright,head := delHeadChar(right)
3854                         right = xright
3855                         line += head
3856                     }
3857                     redraw(lno,line,right);
3858                     continue;
3859                 case EOF:
3860                     goto EXIT;
3861                 case 'R'-0x40: // replace
3862                     dst := convs(line+right);
3863                     line = dst
3864                     right = ""
3865                     redraw(lno,line,right);
3866                     continue;
3867                 case 'T'-0x40: // just show the result
3868                     readDic();
3869                     romkanmode = !romkanmode;
3870                     redraw(lno,line,right);
3871                     continue;
3872                 case 'L'-0x40:
3873                     redraw(lno,line,right);
3874                     continue

```

```

3875         case 'K'-0x40:
3876             right = ""
3877             redraw(lno,line,right);
3878             continue;
3879         case 'E'-0x40:
3880             line += right
3881             right = ""
3882             redraw(lno,line,right);
3883             continue;
3884         case 'A'-0x40:
3885             right = line + right
3886             line = ""
3887             redraw(lno,line,right);
3888             continue;
3889         case 'U'-0x40:
3890             line = ""
3891             right = ""
3892             clearline();
3893             redraw(lno,line,right);
3894             continue;
3895         case 0x7F: // DEL
3896             if( 0 < len(line) ){
3897                 line,_ = delTailChar(line)
3898                 redraw(lno,line,right);
3899             }
3900             continue;
3901         case 'H'-0x40:
3902             if( 0 < len(line) ){
3903                 line,_ = delTailChar(line)
3904                 redraw(lno,line,right);
3905             }
3906             continue;
3907     }
3908     if( ch == '\n' || ch == '\r' ){
3909         fputc(ch,stderr);
3910         break;
3911     }
3912     line += string(ch);
3913     redraw(lno,line,right);
3914 }
3915 EXIT:
3916 system("/bin/stty echo sane");
3917 }
3918 //fprintf(stderr,"\r\nLINE:%s\r\n",line);
3919
3920 EXIT_GOT:
3921     return line + right;
3922 }
3923
3924 func getline_main(){
3925     line := xgetline(0,"",nil)
3926     fprintf(stderr,"%s\n",line);
3927 /*
3928     dp = strpbrk(line,"\r\n");
3929     if( dp != NULL ){
3930         *dp = 0;
3931     }
3932
3933     if( 0 ){
3934         fprintf(stderr,"\n%d\n",int(strlen(line)));
3935     }
3936     if( lseek(3,0,0) == 0 ){
3937         if( romkanmode ){
3938             var buf [8*1024]byte;
3939             convs(line,buf);
3940             strcpy(line,buf);
3941         }
3942         write(3,line,strlen(line));
3943         ftruncate(3,lseek(3,0,SEEK_CUR));
3944         //fprintf(stderr,"outsize=%d\n",int)lseek(3,0,SEEK_END));
3945         lseek(3,0,SEEK_SET);
3946         close(3);
3947     }else{
3948         fprintf(stderr,"\r\n gotline: ");
3949         trans(line);
3950         //printf("%s\n",line);
3951         printf("\n");
3952     }
3953 */
3954 }
3955 //== end ===== getline
3956
3957 //
3958 // $USERHOME/.gsh/
3959 // gsh-rc.txt, or gsh-configure.txt
3960 // gsh-history.txt
3961 // gsh-aliases.txt // should be conditional?
3962 //
3963 func (gshCtx *GshContext)gshSetupHomedir()(bool) {
3964     homedir,found := userHomeDir()
3965     if !found {
3966         fmt.Printf("--E-- You have no UserHomeDir\n")
3967         return true
3968     }
3969     gshhome := homedir + "/" + GSH_HOME
3970     _, err2 := os.Stat(gshhome)
3971     if err2 != nil {
3972         err3 := os.Mkdir(gshhome,0700)
3973         if err3 != nil {
3974             fmt.Printf("--E-- Could not Create %s (%s)\n",
3975                 gshhome,err3)
3976             return true
3977         }
3978         fmt.Printf("--I-- Created %s\n",gshhome)
3979     }
3980     gshCtx.GshHomeDir = gshhome
3981     return false
3982 }
3983 func setupGshContext()(GshContext,bool){
3984     gshPA := syscall.ProcAttr {
3985         "", // the starting directory
3986         os.Environ(), // environ[]
3987         []uintptr{os.Stdin.Fd(),os.Stdout.Fd(),os.Stderr.Fd()},
3988         nil, // OS specific
3989     }
3990     cwd,_ := os.Getwd()
3991     gshCtx := GshContext {
3992         cwd, // StartDir
3993         "", // GetLine
3994         []GChdirHistory { {cwd,time.Now(),0} }, // ChdirHistory
3995         gshPA,
3996         []GCommandHistory{}, //something for invokation?
3997         GCommandHistory{}, // CmdCurrent
3998         false,
3999         []int{},

```



```

4000     syscall.Rusage{},
4001     "", // GshHomeDir
4002     Ttyid(),
4003     false,
4004     false,
4005     []PluginInfo{},
4006     []string{,
4007     " ",
4008     "v",
4009     ValueStack{,
4010     GServer{"", ""}, // LastServer
4011     " ", // RSERVER
4012     cwd, // RWD
4013     CheckSum{,
4014     }
4015     err := gshCtx.gshSetupHomedir()
4016     return gshCtx, err
4017 }
4018 func (gsh *GshContext)gshellh(gline string)(bool){
4019     ghist := gsh.CmdCurrent
4020     ghist.WorkDir,_ = os.Getwd()
4021     ghist.WorkDirX = len(gsh.ChdirHistory)-1
4022     //fmt.Printf("--D--ChdirHistory(@@)\n",len(gsh.ChdirHistory))
4023     ghist.StartAt = time.Now()
4024     rusagev1 := Getrusagev()
4025     gsh.CmdCurrent.FoundFile = []string{
4026     fin := gsh.tgshellh(gline)
4027     rusagev2 := Getrusagev()
4028     ghist.Rusagev = RusageSubv(rusagev2,rusagev1)
4029     ghist.EndAt = time.Now()
4030     ghist.CmdLine = gline
4031     ghist.FoundFile = gsh.CmdCurrent.FoundFile
4032
4033     /* record it but not show in list by default
4034     if len(gline) == 0 {
4035         continue
4036     }
4037     if gline == "hi" || gline == "history" { // don't record it
4038         continue
4039     }
4040     */
4041     gsh.CommandHistory = append(gsh.CommandHistory, ghist)
4042     return fin
4043 }
4044 // <a name="main">Main loop</a>
4045 func script(gshCtxGiven *GshContext) (_ GshContext) {
4046     gshCtxBuf,err0 := setupGshContext()
4047     if err0 {
4048         return gshCtxBuf;
4049     }
4050     gshCtx := &gshCtxBuf
4051
4052     //fmt.Printf("--I-- GSH_HOME=%s\n",gshCtx.GshHomeDir)
4053     //resmap()
4054
4055     /*
4056     if false {
4057         gsh_getlinev, with_exgetline :=
4058             which("PATH",[]string{"which","gsh-getline","-s"})
4059         if with_exgetline {
4060             gsh_getlinev[0] = toFullpath(gsh_getlinev[0])
4061             gshCtx.GetLine = toFullpath(gsh_getlinev[0])
4062         }else{
4063             fmt.Printf("--W-- No gsh-getline found. Using internal getline.\n");
4064         }
4065     }
4066     */
4067
4068     ghist0 := gshCtx.CmdCurrent // something special, or gshrc script, or permanent history
4069     gshCtx.CommandHistory = append(gshCtx.CommandHistory,ghist0)
4070
4071     prevline := ""
4072     skipping := false
4073     for hix := len(gshCtx.CommandHistory); ; {
4074         gline := gshCtx.getline(hix,skipping,prevline)
4075         if skipping {
4076             if strings.Index(gline,"fi") == 0 {
4077                 fmt.Printf("fi\n");
4078                 skipping = false;
4079             }else{
4080                 //fmt.Printf("%s\n",gline);
4081             }
4082             continue
4083         }
4084         if strings.Index(gline,"if") == 0 {
4085             //fmt.Printf("--D-- if start: %s\n",gline);
4086             skipping = true;
4087             continue
4088         }
4089         if false {
4090             os.Stdout.Write([]byte("gotline:"))
4091             os.Stdout.Write([]byte(gline))
4092             os.Stdout.Write([]byte("\n"))
4093         }
4094         gline = strsubst(gshCtx,gline,true)
4095         if false {
4096             fmt.Printf("fmt.Printf %%v - %v\n",gline)
4097             fmt.Printf("fmt.Printf %%s - %s\n",gline)
4098             fmt.Printf("fmt.Printf %%x - %x\n",gline)
4099             fmt.Printf("fmt.Printf %%U - %s\n",gline)
4100             fmt.Printf("Stoutt.Write -")
4101             os.Stdout.Write([]byte(gline))
4102             fmt.Printf("\n")
4103         }
4104         /*
4105         // should be cared in substitution ?
4106         if 0 < len(gline) && gline[0] == '!' {
4107             xgline, set, err := searchHistory(gshCtx,gline)
4108             if err {
4109                 continue
4110             }
4111             if set {
4112                 // set the line in command line editor
4113             }
4114             gline = xgline
4115         }
4116         */
4117         fin := gshCtx.gshellh(gline)
4118         if fin {
4119             break;
4120         }
4121         prevline = gline;
4122         hix++;
4123     }
4124     return *gshCtx

```

```

4125 }
4126 func main() {
4127     gshCtxBuf := GshContext{}
4128     gsh := *gshCtxBuf
4129     argv := os.Args
4130     if 1 < len(argv) {
4131         if isin("version",argv){
4132             gsh.showVersion(argv)
4133             return
4134         }
4135         comx := isinX("-c",argv)
4136         if 0 < comx {
4137             gshCtxBuf,err := setupGshContext()
4138             gsh := *gshCtxBuf
4139             if !err {
4140                 gsh.gshellv(argv[comx+1:])
4141             }
4142             return
4143         }
4144     }
4145     if 1 < len(argv) && isin("-s",argv) {
4146     }else{
4147         gsh.showVersion(append(argv,[jstring{"-l","-a"}...]))
4148     }
4149     script(nil)
4150     //gshCtx := script(nil)
4151     //gshell(gshCtx,"time")
4152 }
4153 //</div></details>
4154 //<details id="gsh-todo"><summary>Consideration</summary><div class="gsh-src">
4155 // - inter gsh communication, possibly running in remote hosts -- to be remote shell
4156 // - merged histories of multiple parallel gsh sessions
4157 // - alias as a function or macro
4158 // - instant alias end environ export to the permanent > ~/.gsh/gsh-alias and gsh-environ
4159 // - retrieval PATH of files by its type
4160 // - gsh as an IME with completion using history and file names as dictionaies
4161 // - gsh a scheduler in precise time of within a millisecond
4162 // - all commands have its subucomand after "---" symbol
4163 // - filename expansion by "-find" command
4164 // - history of ext code and output of each commoand
4165 // - "script" output for each command by pty-tee or telnet-tee
4166 // - $BULLTIN command in PATH to show the priority
4167 // - "?" symbol in the command (not as in arguments) shows help request
4168 // - searching command with wild card like: which ssh-*
4169 // - longformat prompt after long idle time (should dismiss by BS)
4170 // - customizing by building plugin and dynamically linking it
4171 // - generating syntactic element like "if" by macro expansion (like CPP) >> alias
4172 // - "!" symbol should be used for negation, don't wast it just for job control
4173 // - don't put too long output to tty, record it into GSH_HOME/session-id/comand-id.log
4174 // - making canonical form of command at the start adding quotation or white spaces
4175 // - name(a,b,c) ... use "(" and ")" to show both delimiter and realm
4176 // - name? or name! might be useful
4177 // - htar format - packing directory contents into a single html file using data scheme
4178 // - filepath substitution shold be done by each command, especially in case of builtins
4179 // - @N substitution for the history of working directory, and @spec for more generic ones
4180 // - @dir prefix to do the command at there, that means like (chdir @dir; command)
4181 // - GSH_PATH for plugins
4182 // - standard command output: list of data with name, size, resouce usage, modified time
4183 // - generic sort key option -nm name, -sz size, -ru rusage, -ts start-time, -tm mod-time
4184 // -wc word-count, -grep match line count, ...
4185 // - standard command execution result: a list of string, -tm, -ts, -ru, -sz, ...
4186 // - -tailf-filename like tail -f filename, repeat close and open before read
4187 // - max. size and max. duration and timeout of (generated) data transfer
4188 // - auto. numbering, aliasing, IME completion of file name (especially rm of quieer name)
4189 // - IME "?" at the top of the command line means searching history
4190 // - IME %d/0x10000/ %x/ffff/
4191 // - IME ESC to go the edit mode like in vi, and use :command as :s/x/y/g to edit history
4192 // - gsh in WebAssembly
4193 // - gsh as a HTTP server of online-manual
4194 //---END--- (^-^)/ITS more</div></details>
4195 /*
4196 <details id="references"><summary>References</summary><div class="gsh-src">
4197 <p>
4198 <a href="https://golang.org">The Go Programming Language</a>
4199 <iframe src="https://golang.org" width="100%" height="300"></iframe>
4200
4201 <a href="https://developer.mozilla.org/ja/docs/Web">MDN web docs</a>
4202 <a href="https://developer.mozilla.org/ja/docs/Web/HTML/Element">HTML</a>
4203 CSS:
4204 <a href="https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Selectors">Selectors</a>
4205 <a href="https://developer.mozilla.org/en-US/docs/Web/CSS/background-repeat">repeat</a>
4206 HTTP
4207 JavaScript:
4208 ...
4209 </p>
4210 </div></details>
4211 */
4212 /*
4213 <details id="html-src" onclick="frame_open();"><summary>Total Source of GShell</summary><div>
4214
4215 <h2>The full of this HTML including the Go code is here.</h2>
4216 <details><summary>Whole file</summary>
4217 <span id="src-frame"></span></-- a window to show source code -->
4218 </details>
4219 <details onclick="fill_CSSView()"><summary>CSS part</summary>
4220 <span id="gsh-style-view"></span>
4221 </details>
4222 <details onclick="fill_JavaScriptView()"><summary>JavaScript part</summary>
4223 <span id="gsh-javascript-view"></span>
4224 </details>
4225 <details onclick="fill_DataView()"><summary>Builtin data part</summary>
4226 <span id="gsh-data-view"></span>
4227 </details>
4228
4229 </div></details>
4230 */
4231 /*
4232 <div id="gsh-footer" style=""></div><!-- ----- END-OF-VISIBLE-PART ----- -->
4233
4234
4235 <style id="gsh-style-def">
4236 #gsh {border-width:1;margin:0;padding:0;}
4237 #gsh {font-family:monospace,Courier New;color:#ddf;font-size:8px;}
4238 #gsh header{height:100px;}
4239 #gsh header{height:100px;background-image:url(GShell-Logo00.png);}
4240 #gsh-menu{font-size:14pt;color:#f88;}
4241 #gsh-footer{height:100px;background-size:80px;background-repeat:no-repeat;}
4242 #gsh note{color:#000;font-size:10pt;}
4243 #gsh h2{color:#24a;font-family:Georgia;font-size:18pt;}
4244 #gsh details{color:#888;background-color:#aaa;font-family:monospace;}
4245 #gsh summary{font-size:16pt;color:#24a;background-color:#eef;height:30px;}
4246 #gsh pre{font-size:11pt;color:#223;background-color:#faffff;}
4247 #gsh a{color:#24a;}
4248 #gsh a[name]{color:#24a;font-size:16pt;}
4249 #gsh .gsh-src{white-space:pre;font-family:monospace,Courier New;font-size:11pt;}

```

```

4250 #gsh .gsh-src{background-color:#faffff;color:#223;}
4251 #gsh-src-src(spellcheck:false)
4252 #src-frame-textarea(white-space:pre;font-family:monospace,Courier New;font-size:11pt;}
4253 #src-frame-textarea{background-color:#faffff;color:#223;}
4254 .gsh-code {white-space:pre;font-family:monospace !important;}
4255 .gsh-code {color:#088;font-size:11pt; background-color:#eef;}
4256 #gsh-winId {color:#000;font-size:14pt;}
4257 @media print {
4258 #gsh pre(font-size:11pt !important;)
4259 }
4260 }
4261 </style>
4262 <!--
4263 // Logo image should be drawn by JavaScript from a meta-font.
4264 // CSS seems not follow line-splitted URL
4265 -->
4266 <script id="gsh-data">
4267 //GshLogo="OR-ITS-more.jp.png"
4268 GshLogo="data:image/png;base64,\
4269 iVBOw0KGoAAANSUHEUGAAEQEAAAB/CAYAAADv3f4AAAAAAXNSR0Iars4c6QAAAHlWELm\
4270 TUOAgAAAQABAEaAAUAAAABAAAPgEBAAUAAAABAAAARgEoAAUAAAABAAIAAIadpaQAAAAB\
4271 AAAATgAAAAAAABIAAAAQAAAEqAAABAAQoQADAAAQAABAAcQgAEAAAQAAQAAQCGwAE\
4272 AAAAQAAAH8AAAARy1BhgAAAlwSF1zAAALeWACkMBAJgcGAAAF3JRJREUeAhtnQuUFNWZ\
4273 x+t+Tuk23iCgg0/jY6osb8WgvMznV7uG4+b1STR7YnXQdQPCkCj2aWnLD2MS1RkeUAvnoCdu\
4274 4iUJx7jrIY750D0GmF2VqIBEiSggCoMMA+mu+vu//ZMD9U1daU6a2aUby91GKRj3vvd6/q\
4275 fnXvdX8tBA8SIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4276 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4277 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4278 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
4279 2eXs9H+ftSKSdHxsc2qgdE7YusS+1qaalKfnY5YsoKHwEPTdK4MQf+5UEXlbLYsaU15\
4280 npDiLkXZCLPIrM53JSUaq98scqU6+i2kK3StuOnY5reBGKJ7Qw7mOVkEC2Toq0iZw0lJhFS\
4281 jBoVHCstMRb3USXEJ8hF7DsmdPb2+u4vWVWFvXbBpMeZUIAE/hcKOGab66EK0GlyNkh56PC\
4282 HxH2VVKORkqh3qUeKi1YdaOfONJ56OkdI6w5BwomOQlyPziON9DLmXpFK/60p2P/Piyov\
4283 N8mfM+/nJWNGjw9KqOTOVGSF2+2pRi11gn31j0V7YsoVVMzEuVPFRKYdf0ak2LRSB0q\
4284 zrWocCOG6EhwGracJ/dktj3g7dXxH4gKN6AS02pZyergS6AozDQqf79SKTRXHu/+9FN\
4285 L66as8pU/Pn1Pn1TLQJKS73dPXSr20ur7iIwPC8QhbNnCuhYU1lryOTQvYF5.fvqBL7jX\
4286 +cNHjBj5gRydlJHy39o84D4H02Lx8THaPeFUIOU+wLc+KnyhK5FGEVOWGAExB83eMoLY\
4287 rIkbD9gHEP52Vq014h89FUa6kYfYfbbQbnzLJqz4FieNDHCwVUoeiVQOB/5C9PY9DlUaeOH\
4288 +zghUhnSgOqrm0WgukrI9RpjBD4Y6uqcQd5TUOM63z3DMHesy14V491sbdKyxhClCPFR\
4289 Uj6toACF7F9V58NBEDHTOMBaE74Ent+eWrrWr+L/QZw60AdB7QUJjs/OA7COoNBCEMuZ\
4290 ttCu/coG28ftPvKELTPFV8juRasEahhVxaRlguoeBPYfUdo4+OfEBdybL84z9XeSXPAMOC\
4291 bgGvov01zgGw4JF392xnHhdca+Wf3JTjntZ2yClYJBXJNUU5KIKYck1sXRdld68mceVW\
4292 aJovy/VBacMeqvEP46/zlnJjt9jxl7VL53Z15MtvapLQCLNHw5pQDqKyNTQ2z8bnCG2VZ\
4293 qOQ7JsdYvV0AZ2Fayidv6FJ35CS4jXZ9hIR7e27zmpj3T8hLJpYkCjPv1HtK/DJFU4Jw1\
4294 1ImhXN5IR9fazzGRKx4w/C+HQSPEx+rIyrN3qEPtNahsHALD2xh50SCoPPVdEgqcm/8e\
4295 7/dz0AhtpAg/mLKJ77U0VGOxybTdX/Ex/Ptfa/i7r7KucsoiCxiUrohtUXF16wE9H+ccVg1\
4296 pd/CFU42AK2IUP1vTK1L/sJjy5SPvHer728NzvfUzVvDODGy9GoopuuhNLLfCx48HLLZGH\
4297 f/8hPvU/43rQ9xtq6YteV1KDC3fmdQ9nbnf21e7wKE1bOK65icbu0Eqhd3IAw82wdKPUV\
4298 hrauc6ZwKdeJ2x0DRgVQF150xwH6ovJRKAAM46pVt+RxAJVLjW7vY9/+CeUBMK168/rPQn\
4299 WNW4+Py9jXupEdL/HXNZTsvsesLD2vWHWt9mu5rvvXZ9foS4v/LfmgdEpHdGLf2uCW\
4300 Jjly2WENPz23FciVd+zZNCNjYtrNyhyGao8RoJATmRiqoCJnrW5FPtN++fRtwd4S1U2\
4301 bV1WbfffLcRF04qazRD7176/rBjKyLd5pBiZ5wi4Qu7tikPBeCOpuW+kj0sqP8GHNaUzLw\
4302 IOzuywDh9z2R2x0DRgVQF150xwH6ovJRKAAM46pVt+RxAJVLjW7vY9/+CeUBMK168/rPQn\
4303 mCufKzalDFN/y18gA5iwC3dkKhysvZuCVSvG/KheWfPWRDKAMMcD8EKX+rHF12A9bt2d172\
4304 2CufOvzCDymFtNny7QogXDXWIKAIQ7COQZchyADWnerqN5vXttc3sdq2OtrwgmUJ7A+EH7\
4305 yhYbUgmlX7f7K1DwaRyUFN42F1UzNddVEtamL65CY9R26vtbZaw2px8Nfmehz3EM+mgso1k\
4306 d3/ZnBGEL1KPGUWZxg1Yc5ew5/zBzY54awOgwKfNfbwqptcevtW4FUBvov32gew8LDZMTjA\
4307 augg7t/bMXx+yw/egJGKOtKsy2d+gFBb9vDvX5b12TOR+wfjyb0p6UOXGOYNgR/guta3vB\
4308 Fvea6qv2d7v8fdv3rldBw34GSPg91ODG9h5XWkh9kaAMyJ6k1PzZmtD3cnu7vvt50\
4309 h/YrGlP7Wxp/VvuRduc+wsg54ym+8zXKogYSPRPA4IKoGz118b6ytaqCEmb9v/m09CUATZ\
4310 Jow6VnPCMKHzj+sNnpHsCjYja6cArRMyrGkiwF415UoiuLiLRW7fmlLX3z2+gfW1L2\
4311 Y72b6EazkfyOetJ1J50lnJyLdrFUrZUP/3pmkug/yN9AgoGYWTF7neVix/6CHUghlluh/\
4312 f9Uov+gG703q7zFL8XQ+zW+/8FPW6fv7XsXhInlayWdz2X1ULM/4uLmPwNoA5ucdoL3\
4313 ZF6acgoshZT6G041NR5Doj9xuv1cy+rFbcujVsnLkVOCefphUbICLRMv1+9PK4vngHq6Fc2\
4314 NCGMSiCancKfXed+mtflBwuxdmF0zqt/194225Y3TCzrPzQWhtHG2zHraJo/y0khdpanZq\
4315 KxWF66/8CbsAhhcbzdpnhUjeC6YFowJgZemZtqNCDeKtXiVuc3Lk4yVTJepuq5tqSfWKkXA\
4316 ufU9mFwIG3sqnntcX76+3xEXQWwZveqSvprZmc2afYSVY461+04KvyGicCugG2rPp0yT\
4317 o2Um2JwZeo+F6K0dFtNXfW2U9x70/bqZct5z0Poi0+vdpyDjcdxR34U9XceHr10Skt33c\
4318 AcwtK009PF2Fn+gWwLdS6DcFodrAxneOCfRXWUSoK93bZXNvAe+gwr506/204LXpZngJ\
4319 76HgrdvtH2ZMlMYVqgm5ZTP5+7volRR/zJlOYlx+8ohOzEb+CV/OTU5ic3NGfJDS3OMZ\
4320 tFUtLi+Yi4yFcpk7zqpzyb6HlgJebwpgLYx009/j8k//wW3xS32gQPHrV5amTPl1fms20\
4321 fz5yW4HfMkXD+/Buy4NvU73yEFOBK65icot+zjP+8qf4JkYitnGktB/gST0MKKAC18jJPL\
4322 4APCYXNPkMcJrEV84HpyOsww/BSqyT2RG26zr210gA9sBhEp46hsP2ratmOJeGrugBWB2Bz\
4323 NYD1B4OSTMBmcdS2E/GG2ZvrF7UejsqyW/7A7guEH6Ky19q3fPQvQXt4dz+Ueg+Lmy5V\
4324 bjjYtO+b5LSqg5Nz6nwbFhudaYgemZy4ap1z5dlbY3NQTC4F3RKYfOTkAUF9Xry0Lw8H\
4325 dMCC/H230rvoGTMIvC+izhTua27rgAebk4+8H3P553qOoyu/wHj21Zwbd7z2XLv44f1gmQSV\
4326 42M+6KmhorvaWQne1lyz/gLlX+IBncn2FQ7F9Y5XQFN/gUa+Hr3URAggIMTLR7G3fPvPt\
4327 m645oyCzcJmZ9XnQ2JAggBByMSL9VzSQSgfXU8jHpbXbz+m+KueBRRIotE/Bw8ogf/LIzH\
4328 /9Tcnsh681t7DtgQRE81EVT29eWTSJf71FSz0VlyfTLvqt0b62etccBRO1HeS68SY7\
4329 20a2degwMR7W57ng7dkRv19rLztoPMBK73naAYrdzf+5DzsymDyamaHnClOkvPOVHGSFQs\
4330 cY6RwU9DK5MU9wQXMaX+ePguLw8/dvfg6ULLPvsPBPXpOnIQwagELsm9ggNxtc0E01vj5\
4331 7tBBBjAdhMkMPd0/q/iRw1bf44t5cNKQkwaG7DsuZhl6Cl28bk+1u2u78FXyWfklQ4/q2X\
4332 TyvjX8boyN6z6vc9/Ojwz7pUtv1Lp0N2UXLo8PKODMuluvoottjlyxcrNWHhEj0WsyKrkPs\
4333 2Jh14LpJicQXOyp6Ms5fysKeille0G95+WxcEj3m5mcmjNe5b+1yBZYLXgJrmdN/YHMK0S\
4334 aP7Md34PueUyZ28WDovSjzXF/xsFe+Lpz/wjQQ9eIh94ZwqVS62+CUHv31MtNjsfHxorF\
4335 wKgz9FwIrTCRjwJh5+/ocSLzQ1zG52BvtYg+wOpqXRyEwcaRfrdbSc5bD/PySxhBakPWO\
4336 qz9Y4L1U0AB4Xk5we8qDsH06+bd0nwzjFXyAUUvly6Ece0017SAZx0uqxmtzB9RcaVvxx\
4337 2CBMBjAdTcrWMyKriwy4myTh92t3R93/8X1j0ESwety7qPFIlodwAmhFEA2KD6D1wNe6h\
4338 H2HwWwIALQH0QUYZr6yznTls7rgu4OYBJq4JBWJCayRhTyeYx4X8/xCw+rusL9550A+W\
4339 8v0WONZ2xAW7ADZPCEdpXpsLXoDkefremE+yj47AeAa7YxzmjXm+61FzUL46ch7OoD6Q/m\
4340 Wncf9BTVXbs6z3hXpIvm1kjhJubTFkRbAgLQCMiwbuiiPtyKlHhWZaq8YKocMej191y9LY\
4341 Pwk79U/55Bk75fSXmchwhj79Y35x7qU8YspvTbqSG+55hdjn6Y6SErYqVQOL2XoeLrbmWj\
4342 YwkG5S25p1I0K5djzgs+2LB1B4Z6/gG+uosA6yUwOY1jzcCuoG41lqxVQ0EP1wulXUL4PR\
4343 d33GL6w1VE4J35xekPKNISubB/34RwB6JXGz6rflBBjBhJ7t1WBGDvDb4bieXgpPbhN\
4344 NQT3iqMhz7ETHvURXnv45r8FPfQWRNdIQFV2qBlXEF16+rgDLV82CTNvYBIDbS2fJBwMJp\
4345 aW3rXYbqm9qXmLnChjCnvUN5fKMRC2LbZJBk8mU55cn4x/2rLdJQzjntKkyuu01pdqccfMz\
4346 qKp/ahfXooVi+JtoFimZuJyn8F7QHmAMXdaUeTX6c7F07SUUqgyq50z33vV/z0C7b+scH\
4347 Lt.np1tH3Yw841pGt4JWauW7pn5xwqjxb4IMabC3Q8rFlZPCJfTc0S0f8naDzSfWqyfHBU\
4348 mld1jTHGhn3eSrt+42Mk5KwTsxrMe35R7rvorP3rmm49VMogfP80ID191L61dVbXmkqjvb\
4349 nfydX9m8WimZMLKZeSL/VzQSkDPzcdYcye71q/B4XKfKQaNeK3mL47z29fQL/gaT+/vrEO\
4350 dPTX0U9UwBkUVMh9MYULzjvPzxxu0fPOO/pTedhod/1XXGZawFuXp6GIlz+eme291bo\
4351 0xuU119fObLAKGqOhaFa5VNPVhxjK7X0GLOuMRm+JAFefsnnaKzLRhZXLyBf5edUwKc1/wD7\
4352 fd+JL72vEtDPEIqgWkZj6zEP/d5duzt+ZHihXkLhns7umT01LajkyVScenpJW1AAACzAE\
4353 dq25x/+s+NLN0DeLXVtD/SkUr+JL5/9VsbL75z+bvNS8Q2EuQn/Oa3x1/FJZS/VZ30EGCBg\
4354 ePdtCYCRORCKr36vL0pof7EXVDAaVzcGjOECZ56CYcmxZ/7CyUwAr2IIN2XkAOC075/\
4355 4yMTRk3xuyfEGgmxt/xdbpt8uSRI7F11UoFJtQmJ3u7cKXfygMvSfDvpv99RaeH07FRv\
4356 huL4693pW1Yn+FXOC+Cy0VrIWXzylh/w3n7fiiibreUtTsVURMitjpkWRVnPKZdmD2FciM\
4357 dmL1f6+ew10/651MmCDD2YFE12fyCyg38ARAbQSPGX1sCGUCaCRDOUysZauvczG6z2vTfE\
4358 dlGfLXPfjYjYtHckphR+cN+r76LoLJ1d3d451+sndV9Yr4veCW9g+SrxtX6G/arezLB4WX\
4359 tgzv7Wk4n+Z8F/FzZUKIa3ky5ULmo9CE8N3HgLin5ISrNy32hsXoRnTbMvWmiP927o3\
4360 jQg8vnn35zecqN1YgCmlw2/fwicJoJxyt1e0L0XvRGHMyNZ1/IJtL6W63j5y8j+711dy057\
4361 xLJdJmM+X0F0gtrucgEUTDvIpcfnovNAE2KAeVArG5T3tJbGQT+5rCIU+U1BzXPIpJumpRVP\
4362 4YeuZ9Wp9x1fW/oppuyDp9UNPYih91/XNXovNSd5dGG8C8wms31CzfrCKQUTZSHj+v8gJ\
4363 JV7X83M6WqjL5r6LV668T0ExtH7J/4Cdw24+zFvsJrTt1RkFOOALtznZFD2SD12QrQ\
4364 8YSV88pdsboVhRLDQ6exvrEOjy94g9DQPkC5ZmjJy021LdV7y3zfl8qmsDmoFARTVWFC31\
4365 N1NQGWx1JEvagOMrZ78D2ZvefMcdFCU86nfBB5+Kf1FMRHE6FO0S0AotvM/d8V8km7D\
4366 C58YrseFULvsplpxb79z64erdZNYuNLKileJda1Uak7j0orr315x+YA9CQBDF/Ch7JHdB\
4367 E5sg690KMH9pdrJd6v3vgEYvDdQucS1VM9No/Qapp3K2Lve82WcmJ3k30kx+30R0E8K1wN\
4368 blxhE29JqGLR0eFGKam6n5P9mdGP5bntIkpmc22r7BHskjJp0kMCKtCf/KAM1S0JtXeJK\
4369 v7q+/OmzbnN/5ZOHT3+NPgZn2eyx7uiZJDM9xoytCzBT0ya+vndgQ3URP1jYxbmDoe1/au\
4370 zq4BeyggsLmphdGILKxcmLWskzBgwa940stveB+sf714iK3oi05Mxod+r9/I2Vx80P9Ec3\
4371 xp7QYUxJGTqmcato+NeY/99v3xbh+21bh3cn0t1JdfCZnzkeapsDN/vjgd4XP4Cnb8w9p\
4372 9zduKz2Q3fev05lytqomo30pz9Ec5sHOY+FXFV150r6xr5HkDFMGaAdK3yA09dyfDdj\
4373 ppf5kjqG6grNyi3DfyKI5h1400Kj1azehB9NtWTFBAGv1uIawS2xvTahfsB50dfrpseBaP\
4374 mR1F1X0m8Xm4xnP/fBy6AvG2ftY58KwNo2mMPEF3sgCf3o4UGGSj/wI54BwLfbvWab720b)

```

```

4375 Xx/MwGlf9ZrXPQmMx5CiAfjiHlYxjhsR7BKkMfG8mLT+D3CdJf2qdolVnN3v3d60xw7hyf\
4376 koSVf0pEpkZfegJWQLd70c6dnplH7z10z33h0LHWYJuIReH7Z7ptxeVe9XWH+3Jdasm60\
4377 1EW51G5j8Eaj2NR0adga71eVORZLBSccvC8Z0u5Ue1JbSpkvGHEcucJRKXfLW0VSSU1atTmW\
4378 LaycfxHpswKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIK\
4379 QAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIK\
4380 QAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIK\
4381 QAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIK\
4382 QAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIK\
4383 QAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIK\
4384 m38w0ncAAAAASUVORK5CYII="";
4385
4386 GshIcon="data:image/png;base64,\
4387 iVBORw0KGgoAAAANSUHEUGAAsAAAB/CAYAAABYmYLZAAAAAANSR0IARs4c6QAAAHhLWE1m\
4388 TUOAKGAAAAGABEAAUAAAABAAAAPgEBAUAAAABAAAARgEoAAMAAAABAAAIAAIdpAQAQAAAB\
4389 AAAATGAAAANAABAIAAAQAQAAEgAAABAAOgAQAQAAAAAQAACAAAGAAEAQAAAAQAARyGWAEE\
4390 AAAAQAAAH8AAAARCT6LzWAAA1wSFLzAAALeWAAcMBA7GcGAAADQRJREfUEaHtN09VNUd\
4391 x9/b21z-iYCKI1K1amW1j/jh6BCKstFEFthlIGPrWdet0oqkEunttrW2FG01YtIatinZ0\
4392 amdaQf6jIyOXI7kgglarVw74b3BAOPkbVajj3e3r94WcJpe93csmcobj784kd/ve772+3nf\
4393 fFv+mx89SIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAE\
4394 SIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAE\
4395 XiEu65tAHewad8Imp2wLTXtadyBmzrT+42pzRsrD3peQvpXsMtrrhgMYcN8fewHXFuap+zyH\
4396 ZUASIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAE\
4397 hf7VweE6pYDz+0h6JmAxwzZnN6REVcG54SQXwhL18xtFFcUwGhX7AMRgIKQAIKQAIHIAH/\
4398 Nbgem3098H0o58sa50440Umz+EzEAE/FWHXfj3ofLD/YP80ZNUKRALVAWlEGg4C/BGSE\
4399 rGk0AMb/d3uClJWJsIyVnsIy0KAQ8FeVYnmsYJYR5F3cmUmtL6v/fwdD1QAv4S7EBpZYQ/\
4400 65pV5codz46QoHyziLiZeDtLk7m5Layw2ywTmXNDa8cLcplj1wW0i0U1/3d05692zyGpP\
4401 2DBHZE2Dxp+28XkicYgJg9Fve6Eh5QVCinOVegOkFLiKa7gpKVmBUnKnFoodS8i4tYwAGPT\
4402 e0Lc2e8+3+rApl162LEVYnPa6SQfapwSgmw0Kf8upDWGkzLSBaWPFDreUtyYUeFPWH9e\
4403 xWFfi9QQt4cc27qYoroBVF9CrE/2gnEo/ElFa4DqHaloSUT4rpJzsGLGN5j6Z10OQvRtE\
4404 rHDXjvvnShmYsWPTq6UEZEGEJS2EWmpj4sk38VQA0/zSeLUz5aemLHZIRvkdhpAVH0\
4405 F6R5Zafzf8510gmVortLSeXG/oTLB9s+5h8u0ihusYfz19fGlP2cNSyt1A2//w008aEK\
4406 IX87zhymPTkb3oc1bXxa/DKX3YpoeHh686jGcSEXUGvXALY+CVN0S08Mmi9BUOOH+0iH\
4407 zFN7pHgbb3kCo0J1FO9zR7rGvN3d1QNRTq4570TSlhkYsJsuok6478h1PhRo6i4d4mmU7N\
4408 4t2xw1BPUliB6GoWg2+T9JpFNKn7Uwbrmu5WgpjG7K1380LuLcApeIWLcADaUXgBoclYs4\
4409 1B7b03kUET04panzx0+9Yp0NNANedFQmN4oxSnokzGtn+f0CANUSNm3unJxgA0vhzuc6+4\
4410 CGJpZdufWmGRf0n8BezIJDxvDeHa+vntqfD1T0QH1kcvCeJ9j5VebVqEtoSfv1/ERXV9FE\
4411 74aV8+8Yc/v6wf7XamN0KqN60Ylem/tGqTOA6dXNTz8wCCamh8M0bur18I5BubkDG6C\
4412 Bbhvm2bRm7195Jv11hSPpwYEn9sXhL6Jne71K1+UwQNK2HcmG5M6VJzWLU1Y1T+9TUXe+E\
4413 sB0ngishohW0r/2FW5g6VhWvWnYt4r/Okuk2Wp20ig1xhfsnYgqgYz5tBav0/PdhIDX8F9E\
4414 1fhpLHG6ftfbh7V6CFkUXFVwH0ZYSjzc1TK3TjWR1jecr0HTs1rJCXhIu09B1XfVgkC\
4415 5WpZdRZKXN21lqU08pLmxu0CazPaVgJdel20+SUopx3S1L+3idYU2NxxeCDUVAJk0842E\
4416 CPge4UaP7R/74/WPD77y76+A3c574a/PyENpbyb9Yb/cvZPn3oYfvtv3PCJGPJ0fAKqAet3\
4417 7T4w6jEqpKHy0eaEKuDXfG7FWKkhh72Wx453a+vKbSwTla7r0hprYM9Syt0ldv0fd2/1\
4418 Tr/7DA8W5jK8HWVTLKfUmTs4CszRV4I1a+Y8t/zd1l0now1VcTe7mfDHK3+TazxSTjklI2K6\
4419 C8zVlgeGfK8rMeUxNGQ9UBVok390MFZTUyZa8pA05R9YnejKA0/hu0TnW+cc5Y264acLN\
4420 TyPH0Lr+3pl9VMMidCp6pLlTstasnRjCq+BiH0q9KGNrXmH4drYnEzuJfavZkbtINQKZX\
4421 eQd16tyHYZK8wR2c21h9JeY9+0/wj2AjrQ+hPTPKYUtz0gipvsrX076zobWlYj0eBfN3\
4422 csWYwSFsi3RXTkH17ky7CCT+Geaorst0dzhgMI/09rVwtAhpuzayokf99UCZDBL21lH0gE\
4423 213vxcBgeyZDF3bH6gAT7E3yc0VARndI0xm/886d1mVSB1TZP258ndaChWpHma0MmbfM\
4424 vhdsgJNGjKw80JuB41F/WePc4PAlZg1gtD1zu7VWBWRp9g/ubAB6EYVYL/ulF8EXgvC\
4425 V9G6EnbQ/DSrW0YsRxRiQB34E0x/ssYPD73W9owbTpBzeP++jFTSogyoAzc6xr/ofj5QcDY\
4426 BnasW4zhsv+2rDy5zq7AVdp5We1t/GQSumZV6HgmGfPJRO/y4aaU+7GfYl+L50KkwcC+3\
4427 AjjzxwFLD+Bx307RA6IHtuc8jclEpnNz5zq4PS8X09H9p55d2S3TmJnq8zUPTN+OL/PC\
4428 dc2PAUFhsmfn47H6RP12VnwjzrZ5LufLSLB0YF72KosQJyIzN2fL0aGK4U6b8+XyQ\
4429 TkvWvenYqepTqz2f2H6hg26/jB8aPKnoBo59jZLh9L+084E59USUQhki65VwG6P3njYdW\
4430 85ziR001+qabRR6Tso+rzBmQxwv2xrc0csSSmQI/fcFY7LPdZ21Jzr5Kk+C5dELh6ixYITW\
4431 V1/nm4/cmbcW+XmNwee48EzNelwaOfEKYUroI1D0GpL3PawpZRGfPnIhtCOXyQ5CLqPQW\
4432 RG34fbl+LEuV3VncC8bZf4KVlzeZfVNV56qjsoVw+Y0t29BUzqWrgWZD1i0BJWZxe18V1X\
4433 KEPL9fdsBxp/2Xs6sGxKM3dfClatf8adBN1U0UNh1AEWg6Bw93sevgj1HMULPw/b14a6npq\
4434 pkSWLwr06fYmN+v3M1U6MwfbD3KwyWtYxwJz2Ucu04sJ+6WUyJBTLLlpSv1okE327S/NJX\
4435 MgJ+21W6Wt1T14TY/bUNdXmS71u9baqa20Y5DhUX1z9BRGpEvdRHJ51k3m3z394VgdsYp\
4436 qZbnk1kqbbVhBteH161/0vu/ZgszaeFLR+tNOBCXY90Xa7g7Bbt06tbuV/oyiPhu8hzA4R7\
4437 0lMaOu3EQJZY2HWwct1aI7ndi3bXo2P7v2p70cmEPEycw8L4Q6770Ev+htZPnED+mNfy/W2\
4438 1RATh75EJ/O5gf1lW7StTREMd4gAu5Q3T6ARsQdmX7z+/GB47ui29U0Wv9JXAnJ711S\
4439 16Y+xi8s6mYcrRoxul4Klys8B69E110YHs79i/4cxbvgnH2jWB1j1XxexYQuz20g5WdLuq\
4440 4Yaf+5Q5EXMARKAoI2CCP2xvJNV+LMZ78LkH3V27LWVt2n9w4/+6zgdXJPLq7b1TADkwlP\
4441 nHs+QSI+HiEwSRPvengV20d6Nf7K0tloPldj/1kUsatCEBEIABEIAEIAEIAEIAEIAEIAE\
4442 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4443 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4444 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4445 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4446 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4447 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4448 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4449 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4450 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4451 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4452 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4453 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4454 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4455 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4456 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4457 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4458 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4459 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4460 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4461 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4462 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4463 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4464 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4465 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4466 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4467 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4468 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4469 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4470 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4471 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4472 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4473 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4474 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4475 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4476 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4477 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4478 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4479 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4480 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4481 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4482 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4483 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4484 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4485 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4486 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4487 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4488 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4489 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4490 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4491 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4492 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4493 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4494 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4495 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4496 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4497 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4498 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\
4499 IABEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAEIAE\

```

```

4500     document.getElementById(E_TODO).open=true
4501     document.getElementById('References').open=true
4502 }
4503 }
4504 }
4505 var bannerIsStopping = false
4506 //NOTE: .com/JSREF/prop_style_backgroundposition.asp
4507 function shiftBG(){
4508     bannerIsStopping = !bannerIsStopping
4509     bannerStyle.backgroundPosition = "0 0";
4510 }
4511 // status should be inherited on Window Fork(), so use the status in DOM
4512 function html_stop(e,toggle){
4513     if( toggle ){
4514         if( e.innerHTML == "Stop" ){
4515             bannerIsStopping = true
4516             e.innerHTML = "Start"
4517         }else{
4518             bannerIsStopping = false
4519             e.innerHTML = "Stop"
4520         }
4521     }else{
4522         // update JavaScript variable from DOM status
4523         if( e.innerHTML == "Stop" ){ // shown if it's running
4524             bannerIsStopping = false
4525         }else{
4526             bannerIsStopping = true
4527         }
4528     }
4529 }
4530 html_stop(document.getElementById('gsh-menu-stop'),false) // onInit.
4531 //html_stop(bannerElem(),false) // onInit.
4532 }
4533 //https://www.w3schools.com/jsref/met_win_setinterval.asp
4534 function shiftBanner(){
4535     var now = new Date().getTime();
4536     //console.log("now="+now%10)
4537     if( !bannerIsStopping ){
4538         bannerStyle.backgroundPosition = ((now/10)%100000)+" 0";
4539     }
4540 }
4541 setInterval(shiftBanner,10); // onInit.
4542 }
4543 // <a href="https://developer.mozilla.org/ja/docs/Web/API/Window/open">window.open()</a>
4544 // from embedded html to standalone page
4545 var MyChildren = 0
4546 function html_fork(){
4547     MyChildren += 1
4548     WinId = document.getElementById('gsh-WinId').innerHTML + "." + MyChildren;
4549     newwin = window.open("",WinId,"");
4550     src = document.getElementById('gsh');
4551     newwin.document.write("<"+src.innerHTML);
4552     newwin.document.write("<"+src.innerHTML);
4553     newwin.document.write(src.innerHTML);
4554     newwin.document.write("<"+src.innerHTML); // gsh span
4555     newwin.document.getElementById('gsh-menu-exit').innerHTML = "Close";
4556     newwin.document.getElementById('gsh-WinId').innerHTML = WinId;
4557     newwin.document.close();
4558     newwin.focus();
4559 }
4560 function html_close(){
4561     window.close()
4562 }
4563 function win_jump(win){
4564     //win = window.top;
4565     win = window.opener; // https://developer.mozilla.org/ja/docs/Web/API/window.opener
4566     if( win == null ){
4567         console.log("jump to window.opener("+win+") (Error)\n")
4568     }else{
4569         console.log("jump to window.opener("+win+)\n")
4570         win.focus();
4571     }
4572 }
4573 }
4574 // source code viewr
4575 function frame_close(){
4576     srcframe = document.getElementById("src-frame");
4577     srcframe.innerHTML = "";
4578     //srcframe.style.cols = 1;
4579     srcframe.style.rows = 1;
4580     srcframe.style.height = 0;
4581     srcframe.style.display = false;
4582     src = document.getElementById("src-frame-textarea");
4583     src.innerHTML = ""
4584     //src.cols = 0
4585     src.rows = 0
4586     src.display = false
4587     //alert("--closed--")
4588 }
4589 //<!-- | <span onclick="html_view();">Source</span> -->
4590 //<!-- | <span onclick="frame_close();">SourceClose</span> -->
4591 //<!-- | <span>Download</span> -->
4592 function frame_open(){
4593     oldsrc = document.getElementById("GENSRC");
4594     if( oldsrc != null ){
4595         //alert("--I--(erasing old text)")
4596         oldsrc.innerHTML = "";
4597         return
4598     }else{
4599         //alert("--I--(no old text)")
4600     }
4601     banner = document.getElementById('gsh-banner').style.backgroundImage;
4602     footer = document.getElementById('gsh-footer').style.backgroundImage;
4603     document.getElementById('gsh-banner').style.backgroundImage = "";
4604     document.getElementById('gsh-banner').style.backgroundPosition = "";
4605     document.getElementById('gsh-footer').style.backgroundImage = "";
4606 }
4607 src = document.getElementById("gsh");
4608 srcframe = document.getElementById("src-frame");
4609 srcframe.innerHTML = ""
4610 + "<"+cite id="GENSRC">\n"
4611 + "<"+style>\n"
4612 + "#GENSRC textarea{tab-size:4;}\n"
4613 + "#GENSRC textarea(-o-tab-size:4;)\n"
4614 + "#GENSRC textarea(-moz-tab-size:4;)\n"
4615 + "#GENSRC textarea(spellcheck:false;)\n"
4616 + "<"+style>\n"
4617 + "<"+textarea id="src-frame-textarea" cols=100 rows=20 class="gsh-code">"
4618 + /*<"+html>\n" // lost preamble text
4619 + "<"+span id="gsh">" // lost preamble text
4620 + src.innerHTML
4621 + "<"+/span>"+/html>\n" // lost trail text
4622 + "<"+textarea>\n"
4623 + "<"+cite><!-- GENSRC -->\n";
4624 }

```

```
4625 //srcframe.style.cols = 80;
4626 //srcframe.style.rows = 80;
4627
4628 document.getElementById('gsh-banner').style.backgroundImage = banner;
4629 document.getElementById('gsh-footer').style.backgroundImage = footer;
4630 }
4631 function fill_CSSView(){
4632     part = document.getElementById('gsh-style-def')
4633     view = document.getElementById('gsh-style-view')
4634     view.innerHTML = ""
4635     + "<"+textarea cols=100 rows=20 class="gsh-code">'
4636     + part.innerHTML
4637     + "<"+/textarea>"
4638 }
4639 function fill_JavaScriptView(){
4640     part = document.getElementById('gsh-script')
4641     view = document.getElementById('gsh-javascript-view')
4642     view.innerHTML = ""
4643     + "<"+textarea cols=100 rows=20 class="gsh-code">'
4644     + part.innerHTML
4645     + "<"+/textarea>"
4646 }
4647 function fill_DataView(){
4648     part = document.getElementById('gsh-data')
4649     view = document.getElementById('gsh-data-view')
4650     view.innerHTML = ""
4651     + "<"+textarea cols=100 rows=20 class="gsh-code">'
4652     + part.innerHTML
4653     + "<"+/textarea>"
4654 }
4655 function html_view(){
4656     html_stop();
4657
4658     banner = document.getElementById('gsh-banner').style.backgroundImage;
4659     footer = document.getElementById('gsh-footer').style.backgroundImage;
4660     document.getElementById('gsh-banner').style.backgroundImage = "";
4661     document.getElementById('gsh-banner').style.backgroundPosition = "";
4662     document.getElementById('gsh-footer').style.backgroundImage = "";
4663
4664     //srcwin = window.open("", "CodeView2", "");
4665     srcwin = window.open("", "", "");
4666     srcwin.document.write("<span id=\"gsh\">\n");
4667
4668     src = document.getElementById("gsh");
4669     srcwin.document.write("<+style>\n");
4670     srcwin.document.write("textarea{tab-size:4;}\n");
4671     srcwin.document.write("textarea{-o-tab-size:4;}\n");
4672     srcwin.document.write("textarea{-moz-tab-size:4;}\n");
4673     srcwin.document.write("</style>\n");
4674     srcwin.document.write("<h2>\n");
4675     srcwin.document.write("<+span onclick=\"window.close();\n>Close</span> | \n");
4676     //srcwin.document.write("<+span onclick=\"html_stop();\n>Run</span>\n");
4677     srcwin.document.write("</h2>\n");
4678     srcwin.document.write("<textarea id=\"gsh-src-src\" cols=100 rows=60>");
4679     srcwin.document.write("/<+html>\n");
4680     srcwin.document.write("<+span id=\"gsh\">");
4681     srcwin.document.write(src.innerHTML);
4682     srcwin.document.write("<+span><+html>\n");
4683     srcwin.document.write("</+textarea>\n");
4684
4685     document.getElementById('gsh-banner').style.backgroundImage = banner;
4686     document.getElementById('gsh-footer').style.backgroundImage = footer
4687
4688     sty = document.getElementById("gsh-style-def");
4689     srcwin.document.write("<+style>\n");
4690     srcwin.document.write(sty.innerHTML);
4691     srcwin.document.write("<+style>\n");
4692
4693     run = document.getElementById("gsh-script");
4694     srcwin.document.write("<+script>\n");
4695     srcwin.document.write(run.innerHTML);
4696     srcwin.document.write("<+script>\n");
4697
4698     srcwin.document.write("<+span><+html>\n"); // gsh span
4699     srcwin.document.close();
4700     srcwin.focus();
4701 }
4702 </script>
4703 -->
4704 *///<br></span></details></html>
4705
```