

```

1 /*<html>
2 <span id="gsh">
3 <link rel="icon" href="GShell-Logo05icon.png">
4 <meta charset="UTF-8">
5 <meta name="viewport" content="width=device-width, initial-scale=1.0">
6 <title>GShell-0.2.1 by SatoxITS</title>
7 <header id="banner" height="100px" onclick="shiftBG();" style="">
8 <div align="right"><note>GShell version 0.2.1 // 2020-08-25 // 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-menu">
17 | <span onclick="html_new();">NewWindow</span>
18 | <span onclick="html_open();">Unfold</span>
19 | <span onclick="html_fold();">Fold</span>
20 | <span onclick="html_stop();">Stop</span>
21 | <span onclick="html_close();">Close</span>
22 |</span>
23 */
24 /*
25 <details id="html-src" onclick="frame_open();"><summary>Total Source of GShell</summary><div>
26 <h2>The full of this HTML including the Go code is here.</h2>
27 <span id="src-frame"></span> // a window to show source code
28 </div></details>
29 */
30 /*
31 <details id="overview"><summary>Overview</summary><div class="gsh-src">
32 To be written
33 </div>
34 </details>
35 */
36 /*
37 <details id="index">
38 <summary>Go Source Code Index</summary><div class="gsh-src" onclick="document.getElementById('gsh-gocode').open=true;">
39 Implementation
40 Structures
41 <a href="#import">import</a>
42 <a href="#struct">struct</a>
43 Main functions
44 <a href="#comexpansion">str-expansion</a> // macro processor
45 <a href="#finder">finder</a> // builtin find + du
46 <a href="#grep">grep</a> // builtin grep + wc + cksum + ...
47 <a href="#plugin">plugin</a> // plugin commands
48 <a href="#ex_commands">system</a> // external commands
49 <a href="#builtin">builtin</a> // builtin commands
50 <a href="#network">network</a> // socket handler
51 <a href="#remote_sh">remote-sh</a> // remote shell
52 <a href="#redirect">redirect</a> // StdIn/Out redireciton
53 <a href="#history">history</a> // command history
54 <a href="#usage">usage</a> // resource usage
55 <a href="#encode">encode</a> // encode / decode
56 <a href="#IME">IME</a> // command line IME
57 <a href="#getline">getline</a> // line editor
58 <a href="#fscanf">scanf</a> // string decomposer
59 <a href="#interpreter">interpreter</a> // command interpreter
60 <a href="#main">main</a>
61 </div>
62 </details>
63 */
64 //<details id="gsh-gocode">
65 //<summary>Go Source Code</summary><div class="gsh-src" onclick="document.getElementById('gsh-gocode').open=false;">
66 // gsh - Go lang based Shell
67 // (c) 2020 ITS more Co., Ltd.
68 // 2020-0807 created by SatoxITS (sato@its-more.jp)
69
70 package main // gsh main
71 // <a name="import">Imported packages</a> // <a href="https://golang.org/pkg/">Packages</a>
72 import (
73     "fmt" // <a href="https://golang.org/pkg/fmt/">fmt</a>
74     "strings" // <a href="https://golang.org/pkg/strings/">strings</a>
75     "strconv" // <a href="https://golang.org/pkg/strconv/">strconv</a>
76     "sort" // <a href="https://golang.org/pkg/sort/">sort</a>
77     "time" // <a href="https://golang.org/pkg/time/">time</a>
78     "bufio" // <a href="https://golang.org/pkg/bufio/">bufio</a>
79     "io/ioutil" // <a href="https://golang.org/pkg/io/ioutil/">ioutil</a>
80     "os" // <a href="https://golang.org/pkg/os/">os</a>
81     "syscall" // <a href="https://golang.org/pkg/syscall/">syscall</a>
82     "plugin" // <a href="https://golang.org/pkg/plugin/">plugin</a>
83     "net" // <a href="https://golang.org/pkg/net/">net</a>
84     "net/http" // <a href="https://golang.org/pkg/net/http/">http</a>
85     // "html" // <a href="https://golang.org/pkg/html/">html</a>
86     "path/filepath" // <a href="https://golang.org/pkg/path/filepath/">filepath</a>
87     "go/types" // <a href="https://golang.org/pkg/go/types/">types</a>
88     "go/token" // <a href="https://golang.org/pkg/go/token/">token</a>
89     "encoding/base64" // <a href="https://golang.org/pkg/encoding/base64/">base64</a>
90     "unicode/utf8" // <a href="https://golang.org/pkg/unicode/utf8/">utf8</a>
91     // "gshdata" // gshell's logo and source code
92     "hash/crc32" // <a href="https://golang.org/pkg/hash/crc32/">crc32</a>
93 )
94
95 var NAME = "gsh"
96 var AUTHOR = "SatoxITS(^~^)/"
97 var VERSION = "0.2.1"
98 var DATE = "2020-08-25"
99 var LINESIZE = (8*1024)
100 var PATHSEP = ":" // should be ";" in Windows
101 var DIRSEP = "/" // canbe \ in Windows
102 var GSH_HOME = ".gsh" // under home directory
103 var MaxStreamSize = int64(128*1024*1024*1024) // 128GiB is too large?
104 var PROMPT = "> "
105 var GSH_PORT = 9999
106
107 // -xX logging control
108 // --A-- all
109 // --I-- info.
110 // --D-- debug
111 // --T-- time and resource usage
112 // --W-- warning
113 // --E-- error
114 // --F-- fatal error
115 // --Xn-- network
116
117 // <a name="struct">Structures</a>
118 type GCommandHistory struct {
119     Startat time.Time // command line execution started at
120     Endat time.Time // command line execution ended at
121     ResCode int // exit code of (external command)
122     CmdError error // error string
123     OutData *os.File // output of the command
124     FoundFile []string // output - result of ufind

```

```
125     Rusagev      [2]syscall.Rusage // Resource consumption, CPU time or so
126     CmdId       int           // maybe with identified with arguments or impact
127     // redirecton commands should not be the CmdId
128     WorkDir     string        // working directory at start
129     WorkDirX    int           // index in ChdirHistory
130     CmdLine     string        // command line
131 }
132 type GChdirHistory struct {
133     Dir         string
134     Movedat    time.Time
135     CmdIndex   int
136 }
137 type CmdMode struct {
138     BackGround bool
139 }
140 type PluginInfo struct {
141     Spec        *plugin.Plugin
142     Addr        plugin.Symbol
143     Name        string // maybe relative
144     Path        string // this is in Plugin but hidden
145 }
146 type GServer struct {
147     host        string
148     port        string
149 }
150
151 // <a href="https://tools.ietf.org/html/rfc3230">Digest</a>
152 const ( // SumType
153     SUM_ITEMS   = 0x000001 // items count
154     SUM_SIZE    = 0x000002 // data length (simply added)
155     SUM_SIZEHASH = 0x000004 // data length (hashed sequence)
156     SUM_DATEHASH = 0x000008 // date of data (hashed sequence)
157     // also envelope attributes like time stamp can be a part of digest
158     // hashed value of sizes or mod-date of files will be useful to detect changes
159
160     SUM_WORDS   = 0x000010 // word count is a kind of digest
161     SUM_LINES   = 0x000020 // line count is a kind of digest
162     SUM_SUM64   = 0x000040 // simple add of bytes, useful for human too
163
164     SUM_SUM32_BITS = 0x000100 // the number of true bits
165     SUM_SUM32_2BYTE = 0x000200 // 16bits words
166     SUM_SUM32_4BYTE = 0x000400 // 32bits words
167     SUM_SUM32_8BYTE = 0x000800 // 64bits words
168
169     SUM_SUM16_BSD = 0x001000 // UNIXsum -sum -bsd
170     SUM_SUM16_SYSV = 0x002000 // UNIXsum -sum -sysv
171     SUM_UNIXFILE  = 0x004000
172     SUM_CRCIEEE  = 0x008000
173 )
174 type CheckSum struct {
175     Files       int64  // the number of files (or data)
176     Size        int64  // content size
177     Words       int64  // word count
178     Lines       int64  // line count
179     SumType    int
180     Sum64      uint64
181     Crc32Table crc32.Table
182     Crc32Val   uint32
183     Sum16      int
184     Ctime       time.Time
185     Atime       time.Time
186     Mtime       time.Time
187     Start       time.Time
188     Done        time.Time
189     RusageAtStart [2]syscall.Rusage
190     RusageAtEnd  [2]syscall.Rusage
191 }
192 type ValueStack [][]string
193 type GshContext struct {
194     StartDir   string // the current directory at the start
195     GetLine    string // gsh-getline command as a input line editor
196     ChdirHistory []GChdirHistory // the 1st entry is wd at the start
197     gshPA      syscall.ProcAttr
198     CommandHistory []GCommandHistory
199     CmdCurrent GCommandHistory
200     BackGround bool
201     BackGroundJobs []int
202     LastRusage syscall.Rusage
203     GshHomeDir string
204     TerminalId int
205     CmdDtrace  bool // should be [map]
206     CmdTime    bool // should be [map]
207     PluginFuncs []PluginInfo
208     iValues    []string
209     iDelimiter string // field separator of print out
210     iFormat    string // default print format (of integer)
211     iValStack  ValueStack
212     LastServer GServer
213     RSERV      string // [gsh://]host[:port]
214     RWD       string // remote (target, there) working directory
215     lastCheckSum CheckSum
216 }
217
218 func nsleep(ns time.Duration){
219     time.Sleep(ns)
220 }
221 func usleep(ns time.Duration){
222     nsleep(ns*1000)
223 }
224 func msleep(ns time.Duration){
225     nsleep(ns*1000000)
226 }
227 func sleep(ns time.Duration){
228     nsleep(ns*1000000000)
229 }
230
231 func strBegins(str, pat string)(bool){
232     if len(pat) <= len(str){
233         yes := str[0:len(pat)] == pat
234         //fmt.Printf("--D-- strBegins(%v,%v)=%v\n",str,pat,yes)
235         return yes
236     }
237     //fmt.Printf("--D-- strBegins(%v,%v)=%v\n",str,pat,false)
238     return false
239 }
240 func isin(what string, list []string) bool {
241     for _, v := range list {
242         if v == what {
243             return true
244         }
245     }
246     return false
247 }
248 func isinX(what string, list[]string)(int){
249     for i,v := range list {
```

```
250     if v == what {
251         return i
252     }
253 }
254 return -1
255 }
256
257 func env(opts []string) {
258     env := os.Environ()
259     if isn("-s", opts){
260         sort.Slice(env, func(i,j int) bool {
261             return env[i] < env[j]
262         })
263     }
264     for _, v := range env {
265         fmt.Printf("%v\n",v)
266     }
267 }
268
269 // - rewriting should be context dependent
270 // - should postpone until the real point of evaluation
271 // - should rewrite only known notation of symbols
272 func scanInt(str string)(val int,leng int){
273     leng = -1
274     for i,ch := range str {
275         if '0' <= ch && ch <= '9' {
276             leng = i+1
277         }else{
278             break
279         }
280     }
281     if 0 < leng {
282         ival,_ := strconv.Atoi(str[0:leng])
283         return ival,leng
284     }else{
285         return 0,0
286     }
287 }
288 func substHistory(gshCtx *GshContext,str string,i int,rstr string)(leng int,rst string){
289     if len(str[i+1:]) == 0 {
290         return 0,rstr
291     }
292     hi := 0
293     histlen := len(gshCtx.CommandHistory)
294     if str[i+1] == '!' {
295         hi = histlen - 1
296         leng = 1
297     }else{
298         hi,leng = scanInt(str[i+1:])
299         if leng == 0 {
300             return 0,rstr
301         }
302         if hi < 0 {
303             hi = histlen + hi
304         }
305     }
306     if 0 <= hi && hi < histlen {
307         var ext byte
308         if 1 < len(str[i+leng:]) {
309             ext = str[i+leng:][1]
310         }
311         //fmt.Printf("--D-- %v(%c)\n",str[i+leng:],str[i+leng])
312         if ext == 'f' {
313             leng += 1
314             xlist := []string{}
315             list := gshCtx.CommandHistory[hi].FoundFile
316             for _,v := range list {
317                 if list[i] = escapeWhiteSP(v)
318                     xlist = append(xlist,escapeWhiteSP(v))
319             }
320             //rstr += strings.Join(list," ")
321             rstr += strings.Join(xlist," ")
322         }else
323             if ext == 'e' || ext == 'd' {
324                 // !N@.. workdir at the start of the command
325                 leng += 1
326                 rstr += gshCtx.CommandHistory[hi].WorkDir
327             }else{
328                 rstr += gshCtx.CommandHistory[hi].CmdLine
329             }
330         }else{
331             leng = 0
332         }
333     }
334     return leng,rstr
335 }
336 func escapeWhiteSP(str string)(string){
337     if len(str) == 0 {
338         return "\z" // empty, to be ignored
339     }
340     rstr := ""
341     for _,ch := range str {
342         switch ch {
343             case '\\': rstr += "\\\\"
344             case '\n': rstr += "\\n"
345             case '\t': rstr += "\\t"
346             case '\r': rstr += "\\r"
347             case '\n': rstr += "\\n"
348             default: rstr += string(ch)
349         }
350     }
351     return rstr
352 }
353 func unescapeWhiteSP(str string)(string){ // strip original escapes
354     rstr := ""
355     for i := 0; i < len(str); i++ {
356         ch := str[i]
357         if ch == '\\' {
358             if i+1 < len(str) {
359                 switch str[i+1] {
360                     case 'z':
361                         continue;
362                 }
363             }
364             rstr += string(ch)
365         }
366     }
367     return rstr
368 }
369 func unescapeWhiteSPV(strv []string)([]string){ // strip original escapes
370     ustrv := []string{}
371     for _,v := range strv {
372         ustrv = append(ustrv,unescapeWhiteSP(v))
373     }
374     return ustrv
375 }
```

```

375 // <a name="comexpansion">str-expansion</a>
376 // - this should be a macro processor
377 func strsubst(gshCtx *GshContext,str string,histonly bool) string {
378     rbuf := []byte{}
379     if false {
380         //@@U' Unicode should be cared as a character
381         return str
382     }
383     //rstr := ""
384     inEsc := 0 // escape characer mode
385     for i := 0; i < len(str); i++ {
386         //fmt.Printf("--D--Subst %v:%v\n",i,str[i:])
387         ch := str[i]
388         if inEsc == 0 {
389             if ch == '\'' {
390                 //leng,xrstr := substHistory(gshCtx,str,i,rstr)
391                 leng,rs := substHistory(gshCtx,str,i,"")
392                 if 0 < leng {
393                     //_,rs := substHistory(gshCtx,str,i,"")
394                     rbuf = append(rbuf,[]byte(rs)...)
395                     i += leng
396                     //rstr = xrstr
397                     continue
398                 }
399             }
400             switch ch {
401                 case '\\': inEsc = '\\'; continue
402                 //case '%': inEsc = '%'; continue
403                 case '$':
404             }
405         }
406         switch inEsc {
407             case '\\':
408                 switch ch {
409                     case '\\': ch = '\\'
410                     case 's': ch = ','
411                     case 't': ch = '\t'
412                     case 'r': ch = '\r'
413                     case 'n': ch = '\n'
414                     case 'z': inEsc = 0; continue // empty, to be ignored
415                 }
416             inEsc = 0
417         case '%':
418             switch {
419                 case ch == '%': ch = '%'
420                 case ch == 'T':
421                     //rstr = rstr + time.Now().Format(time.Stamp)
422                     rs := time.Now().Format(time.Stamp)
423                     rbuf = append(rbuf,[]byte(rs)...)
424                     inEsc = 0
425                     continue;
426                     default:
427                         // postpone the interpretation
428                         //rstr = rstr + "%" + string(ch)
429                     rbuf = append(rbuf,ch)
430                     inEsc = 0
431                     continue;
432                 }
433             inEsc = 0
434         }
435         //rstr = rstr + string(ch)
436         rbuf = append(rbuf,ch)
437     }
438 }
439 //fmt.Printf("--D--subst(%s)(%s)\n",str,string(rbuf))
440 return string(rbuf)
441 //return rstr
442 }
443 func showFileInfo(path string, opts []string) {
444     if isin("-l",opts) || isin("-ls",opts) {
445         fi, err := os.Stat(path)
446         if err != nil {
447             fmt.Printf("----- ((%v))",err)
448         }else{
449             mod := fi.ModTime()
450             date := mod.Format(time.Stamp)
451             fmt.Printf("%v %8v %s ",fi.Mode(),fi.Size(),date)
452         }
453     }
454     fmt.Printf("%s",path)
455     if isin("-p",opts) {
456         fmt.Printf(" ")
457     }else{
458         if ! isin("-n",opts) {
459             fmt.Println("\n")
460         }
461     }
462 func userHomeDir()(string,bool){
463     /*
464     homedir,_ = os.UserHomeDir() // not implemented in older Golang
465     */
466     homedir,found := os.LookupEnv("HOME")
467     //fmt.Printf("--I-- HOME=%v(%v)\n",homedir,found)
468     if !found {
469         return "/tmp",found
470     }
471     return homedir,found
472 }
473 func toFullPath(path string) (fullpath string) {
474     if path[0] == '/' {
475         return path
476     }
477     pathv := strings.Split(path,DIRSEP)
478     switch {
479         case pathv[0] == ".":
480             pathv[0], _ = os.Getwd()
481         case pathv[0] == "..": // all ones should be interpreted
482             cwd, _ := os.Getwd()
483             ppathv := strings.Split(cwd,DIRSEP)
484             pathv[0] = strings.Join(ppathv,DIRSEP)
485         case pathv[0] == "-":
486             pathv[0],_ = userHomeDir()
487         default:
488             cwd, _ := os.Getwd()
489             pathv[0] = cwd + DIRSEP + pathv[0]
490     }
491     return strings.Join(pathv,DIRSEP)
492 }
493 }
494 func IsRegFile(path string)(bool){
495     fi, err := os.Stat(path)
496     if err == nil {
497         fm := fi.Mode()
498         return fm.IsRegular();
499     }

```

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


```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

```

1250     found, val := getval(arg)
1251     if found {
1252         fmt.Printf("%d",val)
1253     }else{
1254         fmt.Printf("%s",arg)
1255     }
1256 }
1257 if nblend {
1258     fmt.Println("\n");
1259 }
1260 }
1261
1262 func resfile() string {
1263     return "gsh.tmp"
1264 }
1265 //var refF *File
1266 func resmap() {
1267     _, err := os.OpenFile(resfile(), os.O_RDWR|os.O_CREATE, os.ModeAppend)
1268     // https://developpaper.com/solution-to-golang-bad-file-descriptor-problem/
1269     _, err := os.OpenFile(resfile(), os.O_RDWR|os.O_CREATE, 0600)
1270     if err != nil {
1271         fmt.Printf("refF could not open: %s\n",err)
1272     }else{
1273         fmt.Printf("refF opened\n")
1274     }
1275 }
1276
1277 // @@2020-0821
1278 func gshScanArg(str string,strip int)(argv []string){
1279     var si = 0
1280     var sb = 0
1281     var inBracket = 0
1282     var arg1 = make([]byte,LINESIZE)
1283     var ax = 0
1284     debug := false
1285
1286     for ; si < len(str); si++ {
1287         if str[si] != ' ' {
1288             break
1289         }
1290     }
1291     sb = si
1292     for ; si < len(str); si++ {
1293         if sb <= si {
1294             if debug {
1295                 fmt.Printf("--Da- +%d %2d-%2d %s ... %s\n",
1296                         inBracket,sb,si,arg1[0:ax],str[si:])
1297             }
1298             ch := str[si]
1299             if ch == '{' {
1300                 inBracket += 1
1301                 if 0 < strip && inBracket <= strip {
1302                     //fmt.Printf("stripLEV %d <= %d?\n",inBracket,strip)
1303                     continue
1304                 }
1305             }
1306             if 0 < inBracket {
1307                 if ch == ')' {
1308                     inBracket -= 1
1309                     if 0 < strip && inBracket < strip {
1310                         //fmt.Printf("stripLEV %d < %d?\n",inBracket,strip)
1311                         continue
1312                     }
1313                 }
1314                 arg1[ax] = ch
1315                 ax += 1
1316                 continue
1317             }
1318             if str[si] == ' ' {
1319                 argv = append(argv,string(arg1[0:ax]))
1320                 if debug {
1321                     fmt.Printf("--Da- [%v]{%v-%v} %s ... %s\n",
1322                         -1+len(argv),sb,si,str[sb:si],string(str[si:]))
1323                 }
1324                 sb = si+1
1325                 ax = 0
1326                 continue
1327             }
1328             arg1[ax] = ch
1329             ax += 1
1330         }
1331     }
1332     if sb < si {
1333         argv = append(argv,string(arg1[0:ax]))
1334         if debug {
1335             fmt.Printf("--Da- [%v]{%v-%v} %s ... %s\n",
1336                 -1+len(argv),sb,si,string(arg1[0:ax]),string(str[si:]))
1337         }
1338     }
1339     if debug {
1340         fmt.Printf("--Da- %d %s => [%d]%v\n",strip,str,len(argv),argv)
1341     }
1342     return argv
1343 }
1344
1345 // should get stderr (into tmpfile ?) and return
1346 func (gsh*GshContext)Popen(name,mode string)(pin*os.File,pout*os.File,err bool){
1347     var pv = []int{-1,-1}
1348     syscall.Pipe(pv)
1349
1350     xarg := gshScanArg(name,1)
1351     name = strings.Join(xarg, " ")
1352
1353     pin = os.NewFile(uintptr(pv[0]),"StdoutOf-{"+name+"}")
1354     pout = os.NewFile(uintptr(pv[1]),"StdinOf-{"+name+"}")
1355     fidx := 0
1356     dir := "?"
1357     if mode == "r" {
1358         dir = "<"
1359         fidx = 1 // read from the stdout of the process
1360     }else{
1361         dir = ">"
1362         fidx = 0 // write to the stdin of the process
1363     }
1364     gshPA := gsh.gshPA
1365     savfd := gshPA.Files[fidx]
1366
1367     var fd uintptr = 0
1368     if mode == "r" {
1369         fd = pout.Fd()
1370         gshPA.Files[fidx] = pout.Fd()
1371     }else{
1372         fd = pin.Fd()
1373         gshPA.Files[fidx] = pin.Fd()
1374     }

```

```

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

```

```

1500 // should br repeated by "repeat" command
1501 count, _ := strconv.Atoi(argv[1])
1502 fd := gshPA.Files[1] // Stdout
1503 file := os.NewFile(fd,"internalstdout")
1504 fmt.Printf("--I-- Gen. Count%d to [%d]\n",count,file.Fd())
1505 //buf := []byte{}
1506 outdata := "0123 5678 0123 5678 0123 5678 0123 5678\r"
1507 for gi := 0; gi < count; gi++ {
1508     file.WriteString(outdata)
1509 }
1510 //file.WriteString("\n")
1511 fmt.Printf("\n(%d B)\n",count*len(outdata));
1512 //file.Close()
1513 }
1514
1515 //<a name="rexec">Remote Execution</a> // 2020-0820
1516 func Elapsed(from time.Time)(string){
1517     elps := time.Now().Sub(from)
1518     if 1000000000 < elps {
1519         return fmt.Sprintf("[%5d.%02ds]",elps/1000000000,(elps%100000000)/1000000)
1520     }else
1521     if 100000 < elps {
1522         return fmt.Sprintf("[%3d.%03dms]",elps/1000000,(elps%1000000)/1000)
1523     }else{
1524         return fmt.Sprintf("[%3d.%03dus]",elps/1000,(elps%1000))
1525     }
1526 }
1527 func abftime(nanos int64)(string){
1528     if 1000000000 < nanos {
1529         return fmt.Sprintf("%d.%02ds",nanos/1000000000,(nanos%100000000)/1000000)
1530     }else
1531     if 1000000 < nanos {
1532         return fmt.Sprintf("%d.%03dms",nanos/1000000,(nanos%1000000)/1000)
1533     }else{
1534         return fmt.Sprintf("%d.%03dus",nanos/1000,(nanos%1000))
1535     }
1536 }
1537 func abssize(size int64)(string){
1538     fsize := float64(size)
1539     if 1024*1024*1024 < size {
1540         return fmt.Sprintf("%.2fGiB",fsize/(1024*1024*1024))
1541     }else
1542     if 1024*1024 < size {
1543         return fmt.Sprintf("%.3fMiB",fsize/(1024*1024))
1544     }else{
1545         return fmt.Sprintf("%.3fKiB",fsize/1024)
1546     }
1547 }
1548 func absize(size int64)(string){
1549     fsize := float64(size)
1550     if 1024*1024*1024 < size {
1551         return fmt.Sprintf("%.2fGiB",fsize/(1024*1024*1024))
1552     }else
1553     if 1024*1024 < size {
1554         return fmt.Sprintf("%.3fMiB",fsize/(1024*1024))
1555     }else{
1556         return fmt.Sprintf("%.3fKiB",fsize/1024)
1557     }
1558 }
1559 func abbspeed(totalB int64,ns int64)(string){
1560     MBs := (float64(totalB)/1000000) / (float64(ns)/1000000000)
1561     if 1000 <= MBs {
1562         return fmt.Sprintf("%6.3fGB/s",MBs/1000)
1563     }
1564     if 1 <= MBs {
1565         return fmt.Sprintf("%6.3fMB/s",MBs)
1566     }else{
1567         return fmt.Sprintf("%6.3fKB/s",MBs*1000)
1568     }
1569 }
1570 func abspeed(totalB int64,ns time.Duration)(string){
1571     MBs := (float64(totalB)/1000000) / (float64(ns)/1000000000)
1572     if 1000 <= MBs {
1573         return fmt.Sprintf("%6.3fGbps",MBs/1000)
1574     }
1575     if 1 <= MBs {
1576         return fmt.Sprintf("%6.3fMbps",MBs)
1577     }else{
1578         return fmt.Sprintf("%6.3fKbps",MBs*1000)
1579     }
1580 }
1581 func fileRelay(what string,in*os.File,out*os.File,size int64,bsiz int)(wcount int64){
1582     Start := time.Now()
1583     buff := make([]byte,bsiz)
1584     var total int64 = 0
1585     var rem int64 = size
1586     nio := 0
1587     Prev := time.Now()
1588     var PrevSize int64 = 0
1589     fmt.Printf(Elapsed(Start)+"--In- X: %s (%v/%v/%v) START\n",
1590             what,absize(total),size,nio)
1591
1592     for i:= 0; ; i++ {
1593         var len = bsiz
1594         if int(rem) < len {
1595             len = int(rem)
1596         }
1597         Now := time.Now()
1598         Elps := Now.Sub(Prev);
1599         if 1000000000 < Now.Sub(Prev) {
1600             fmt.Printf(Elapsed(Start)+"--In- X: %s (%v/%v/%v) %s\n",
1601                     what,absize(total),size,nio,
1602                     abspeed((total-PrevSize),Elps))
1603             Prev = Now;
1604             PrevSize = total
1605         }
1606         rlen := len
1607         if in != nil {
1608             // should watch the disconnection of out
1609             rcc,err := in.Read(buff[0:rlen])
1610             if err != nil {
1611                 fmt.Printf(Elapsed(Start)+"--En- X: %s read(%v,%v)<%v\n",
1612                         what,rcc,err,in.Name())
1613                 break
1614             }
1615         }
1616         rlen = rcc
1617         if string(buff[0:10]) == "((SoftEOF " {
1618             var ecc int64 = 0
1619             fmt.Sscanf(string(buff),"(SoftEOF %v",&ecc)
1620             fmt.Println(Elapsed(Start)+"--En- X: %s Recv ((SoftEOF %v))/%v\n",
1621                     what,ecc,total)
1622             if ecc == total {
1623                 break
1624             }
1625         }

```

```

1625     }
1626 }
1627
1628 wlen := rlen
1629 if out != nil {
1630     wcc,err := out.Write(buff[0:rlen])
1631     if err != nil {
1632         fmt.Printf(Elapsed(Start)+"-En-- X: %s write(%v,%v)>%v\n",
1633             what,wcc,err,out.Name())
1634         break
1635     }
1636     wlen = wcc
1637 }
1638 if wlen < rlen {
1639     fmt.Printf(Elapsed(Start)+"--En- X: %s incomplete write (%v/%v)\n",
1640             what,wlen,rlen)
1641     break;
1642 }
1643
1644 nio += 1
1645 total += int64(rlen)
1646 rem -= int64(rlen)
1647 if rem <= 0 {
1648     break
1649 }
1650 }
1651 Done := time.Now()
1652 Elps := float64(Done.Sub(Start))/1000000000 //Seconds
1653 TotalMB := float64(total)/1000000 //MB
1654 MBPs := TotalMB / Elps
1655 fmt.Printf(Elapsed(Start)+"--In- X: %s (%v/%v/%v) %v %.3fMB/s\n",
1656             what,total,size,nio,absize(total),MBPs)
1657 return total
1658 }
1659 func tcpPush(clntr *os.File){
1660     // shrink socket buffer and recover
1661     usleep(100);
1662 }
1663 func (gsh*GshContext)RexecServer(argv[]string){
1664     debug := true
1665     Start0 := time.Now()
1666     Start := Start0
1667 // if local == ":" { local = "0.0.0.0:9999" }
1668     local := "0.0.0.0:9999"
1669
1670     if 0 < len(argv) {
1671         if argv[0] == "-s" {
1672             debug = false
1673             argv = argv[1:]
1674         }
1675     if 0 < len(argv) {
1676         argv = argv[1:]
1677     }
1678     port, err := net.ResolveTCPAddr("tcp",local);
1679     if err != nil {
1680         fmt.Printf("--En- S: Address error: %s (%s)\n",local,err)
1681         return
1682     }
1683     fmt.Printf(Elapsed(Start)+"--In- S: Listening at %s...\n",local);
1684     sconn, err := net.ListenTCP("tcp", port)
1685     if err != nil {
1686         fmt.Printf(Elapsed(Start)+"--En- S: Listen error: %s (%s)\n",local,err)
1687         return
1688     }
1689
1690     reqbuf := make([]byte,LINESIZE)
1691     res := ""
1692     for {
1693         fmt.Printf(Elapsed(Start0)+"--In- S: Listening at %s...\n",local);
1694         aconn, err := sconn.AcceptTCP()
1695         Start = time.Now()
1696         if err != nil {
1697             fmt.Printf(Elapsed(Start)+"--En- S: Accept error: %s (%s)\n",local,err)
1698             return
1699         }
1700         clnt, _ := aconn.File()
1701         fd := Clnt.Fd()
1702         ar := aconn.RemoteAddr()
1703         if debug { fmt.Printf(Elapsed(Start0)+"--In- S: Accepted TCP at %s [%d] <- %v\n",
1704             local,fd,ar) }
1705         res = fmt.Sprintf("220 GShell/%s Server\r\n%s",VERSION)
1706         fmt.Fprintf(clnt,"%s",res)
1707         if debug { fmt.Printf(Elapsed(Start)+"--In- S: %s",res) }
1708         count, err := clnt.Read(reqbuf)
1709         if err != nil {
1710             fmt.Printf(Elapsed(Start)+"--En- C: (%v %v) %v",
1711                 count,err,string(reqbuf))
1712         }
1713         req := string(reqbuf[:count])
1714         if debug { fmt.Printf(Elapsed(Start)+"--In- C: %v",string(req)) }
1715         reqv := strings.Split(string(req),"\\r\\n")
1716         cmdv := gshScanArg(reqv[0],0)
1717 //cmdv := strings.Split(reqv[0]," ")
1718         switch cmdv[0] {
1719             case "HELO":
1720                 res = fmt.Sprintf("250 %v",req)
1721             case "GET":
1722                 // download {remotefile|-zN} [localfile]
1723                 var dszie int64 = 32*1024*1024
1724                 var bsize int = 64*1024
1725                 var fname string = ""
1726                 var in *os.File = nil
1727                 var pseudoEOF = false
1728                 if 1 < len(cmdv) {
1729                     fname = cmdv[1]
1730                     if strBegins(fname,"-z") {
1731                         fmt.Sscanf(fname[2:], "%d", &dszie)
1732                     }
1733                     if strBegins(fname,"{") {
1734                         xin,xout,err := gsh.Popen(fname,"r")
1735                         if err {
1736                             if xin {
1737                                 defer xin.Close()
1738                                 defer xout.Close()
1739                                 in = xin
1740                                 dszie = MaxStreamSize
1741                                 pseudoEOF = true
1742                             }
1743                         }
1744                     }
1745                     if err != nil {
1746                         fmt.Printf("--En- GET (%v)\n",err)
1747                     }else{
1748                         defer xin.Close()
1749                     }
1750                 }
1751             }
1752         }
1753     }
1754 }
```

```

1750         in = xin
1751         fi,_ := xin.Stat()
1752         dsize = fi.Size()
1753     }
1754 }
1755 //fmt.Printf(Elapsed(Start)+"--In- GET %v\n",dsize,bsize)
1756 res = fmt.Sprintf("200 %v\r\n",dsize)
1757 fmt.Fprintf(clnt,"%v",res)
1758 tcpPush(clnt); // should be separated as line in receiver
1759 fmt.Printf(Elapsed(Start)+"--In- S: %v",res)
1760 wcount := fileRelay("SendGET",in,clnt,dsize,bsize)
1761 if pseudoEOF {
1762     in.Close() // pipe from the command
1763     // show end of stream data (its size) by OOB?
1764     SoftEOF := fmt.Sprintf("(SoftEOF %v)",wcount)
1765     fmt.Printf(Elapsed(Start)+"--In- S: Send %v\r\n",SoftEOF)
1766
1767     tcpPush(clnt); // to let SoftEOF data appear at the top of received data
1768     fmt.Fprintf(clnt,"%v\r\n",SoftEOF)
1769     tcpPush(clnt); // to let SoftEOF alone in a packet (separate with 200 OK)
1770     // with client generated random?
1771     //fmt.Printf("--In- L: close %v (%v)\n",in.Fd(),in.Name())
1772 }
1773 res = fmt.Sprintf("200 GET done\r\n")
1774 case "PUT":
1775     // upload {srcfile|-zN} [dstfile]
1776     var dsize int64 = 32*1024*1024
1777     var bsize int = 64*1024
1778     var fname string = ""
1779     var out *os.File = nil
1780     if 1 < len(cmdv) { // localfile
1781         fmt.Sscanf(cmdv[1],"%d",&dsize)
1782     }
1783     if 2 < len(cmdv) {
1784         fname = cmdv[2]
1785         if fname == "-" {
1786             // nul dev
1787         }else{
1788             if strBegins(fname,"{") {
1789                 xin,xout,err := gsh.Popen(fname,"w")
1790                 if err {
1791                     if err {
1792                         defer xout.Close()
1793                         out = xout
1794                     }
1795                 }
1796             }else{
1797                 // should write to temporary file
1798                 // should suppress ^C on tty
1799                 xout,err := os.OpenFile(fname,os.O_CREATE|os.O_RDWR|os.O_TRUNC,0600)
1800                 //fmt.Printf("--In- S: open(%v) out(%v) err(%v)\n",fname,xout,err)
1801                 if err != nil {
1802                     fmt.Printf("--En- PUT (%v)\n",err)
1803                 }else{
1804                     out = xout
1805                 }
1806             }
1807         }
1808         fmt.Printf(Elapsed(Start)+"--In- L: open(%v,w) %v (%v)\n",
1809             fname,local,err)
1810     }
1811     fmt.Printf(Elapsed(Start)+"--In- PUT %v (%v)\n",dsize,bsize)
1812     fmt.Printf(Elapsed(Start)+"--In- S: 200 %v OK\r\n",dsize)
1813     fmt.Fprintf(clnt,"200 %v OK\r\n",dsize)
1814     fileRelay("RecvPUT",clnt,out,dsize,bsize)
1815     res = fmt.Sprintf("200 PUT done\r\n")
1816 default:
1817     res = fmt.Sprintf("400 What? %v",req)
1818 }
1819 swcc,serr := clnt.Write([]byte(res))
1820 if serr != nil {
1821     fmt.Printf(Elapsed(Start)+"--In- S: (wc=%v er=%v) %v",swcc,serr,res)
1822 }else{
1823     fmt.Printf(Elapsed(Start)+"--In- S: %v",res)
1824 }
1825 aconn.Close();
1826 clnt.Close();
1827 }
1828 sconn.Close();
1829 }
1830 func (gsh*GshContext)RexecClient(argv[]string)(int,string){
1831     debug := true
1832     Start := time.Now()
1833     if len(argv) == 1 {
1834         return -1,"EmptyARG"
1835     }
1836     argv = argv[1:]
1837     if argv[0] == "-serv" {
1838         gsh.RexecServer(argv[1:])
1839         return 0,"Server"
1840     }
1841     remote := "0.0.0.0:9999"
1842     if argv[0][0] == '@' {
1843         remote = argv[0][1:]
1844         argv = argv[1:]
1845     }
1846     if argv[0] == "-s" {
1847         debug = false
1848         argv = argv[1:]
1849     }
1850     dport, err := net.ResolveTCPAddr("tcp",remote);
1851     if err != nil {
1852         fmt.Printf(Elapsed(Start)+"Address error: %s (%s)\n",remote,err)
1853         return -1,"AddressError"
1854     }
1855     fmt.Printf(Elapsed(Start)+"--In- C: Connecting to %s\n",remote)
1856     serv, err := net.DialTCP("tcp",nil,dport)
1857     if err != nil {
1858         fmt.Printf(Elapsed(Start)+"Connection error: %s (%s)\n",remote,err)
1859         return -1,"CannotConnect"
1860     }
1861     if debug {
1862         al := serv.LocalAddr()
1863         fmt.Printf(Elapsed(Start)+"--In- C: Connected to %v <- %v\n",remote,al)
1864     }
1865     req := ""
1866     res := make([]byte,LINESIZE)
1867     count,err := serv.Read(res)
1868     if err != nil {
1869         fmt.Printf("--En- S: (%d,%v) %v",count,err,string(res))
1870     }
1871     if debug { fmt.Printf(Elapsed(Start)+"--In- S: %v",string(res)) }
1872     if argv[0] == "GET" {
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
20100
20101
20102
20103
20104
20105
20106
20107
20108
20109
20110
20111
20112
20113
20114
20115
20116
20117
20118
20119
20120
20121
20122
20123
20124
20125
20126
20127
20128
20129
20130
20131
20132
20133
20134
20135
20136
20137
20138
20139
20140
20141
20142
20143
20144
20145
20146
20147
20148
20149
20150
20151
20152
20153
20154
20155
20156
20157
20158
20159
20160
20161
20162
20163
20164
20165
20166
20167
20168
20169
20170
20171
20172
20173
20174
20175
20176
20177
20178
20179
20180
20181
20182
20183
20184
20185
20186
20187
20188
20189
20190
20191
20192
20193
20194
20195
20196
20197
20198
20199
20200
20201
20202
20203
20204
20205
20206
20207
20208
20209
202010
202011
202012
202013
202014
202015
202016
202017
202018
202019
202020
202021
202022
202023
202024
202025
202026
202027
202028
202029
202030
202031
202032
202033
202034
202035
202036
202037
202038
202039
202040
202041
202042
202043
202044
202045
202046
202047
202048
202049
202050
202051
202052
202053
202054
202055
202056
202057
202058
202059
202060
202061
202062
202063
202064
202065
202066
202067
202068
202069
202070
202071
202072
202073
202074
202075
202076
202077
202078
202079
202080
202081
202082
202083
202084
202085
202086
202087
202088
202089
202090
202091
202092
202093
202094
202095
202096
202097
202098
202099
2020100
2020101
2020102
2020103
2020104
2020105
2020106
2020107
2020108
2020109
2020110
2020111
2020112
2020113
2020114
2020115
2020116
2020117
2020118
2020119
2020120
2020121
2020122
2020123
2020124
2020125
2020126
2020127
2020128
2020129
2020130
2020131
2020132
2020133
2020134
2020135
2020136
2020137
2020138
2020139
2020140
2020141
2020142
2020143
2020144
2020145
2020146
2020147
2020148
2020149
2020150
2020151
2020152
2020153
2020154
2020155
2020156
2020157
2020158
2020159
2020160
2020161
2020162
2020163
2020164
2020165
2020166
2020167
2020168
2020169
2020170
2020171
2020172
2020173
2020174
2020175
2020176
2020177
2020178
2020179
2020180
2020181
2020182
2020183
2020184
2020185
2020186
2020187
2020188
2020189
2020190
2020191
2020192
2020193
2020194
2020195
2020196
2020197
2020198
2020199
2020200
2020201
2020202
2020203
2020204
2020205
2020206
2020207
2020208
2020209
20202010
20202011
20202012
20202013
20202014
20202015
20202016
20202017
20202018
20202019
20202020
20202021
20202022
20202023
20202024
20202025
20202026
20202027
20202028
20202029
20202030
20202031
20202032
20202033
20202034
20202035
20202036
20202037
20202038
20202039
20202040
20202041
20202042
20202043
20202044
20202045
20202046
20202047
20202048
20202049
20202050
20202051
20202052
20202053
20202054
20202055
20202056
20202057
20202058
20202059
20202060
20202061
20202062
20202063
20202064
20202065
20202066
20202067
20202068
20202069
20202070
20202071
20202072
20202073
20202074
20202075
20202076
20202077
20202078
20202079
20202080
20202081
20202082
20202083
20202084
20202085
20202086
20202087
20202088
20202089
20202090
20202091
20202092
20202093
20202094
20202095
20202096
20202097
20202098
20202099
202020100
202020101
202020102
202020103
202020104
202020105
202020106
202020107
202020108
202020109
202020110
202020111
202020112
202020113
202020114
202020115
202020116
202020117
202020118
202020119
202020120
202020121
202020122
202020123
202020124
202020125
202020126
202020127
202020128
202020129
202020130
202020131
202020132
202020133
202020134
202020135
202020136
202020137
202020138
202020139
202020140
202020141
202020142
202020143
202020144
202020145
202020146
202020147
202020148
202020149
202020150
202020151
202020152
202020153
202020154
202020155
202020156
202020157
202020158
202020159
202020160
202020161
202020162
202020163
202020164
202020165
202020166
202020167
202020168
202020169
202020170
202020171
202020172
202020173
202020174
202020175
202020176
202020177
202020178
202020179
202020180
202020181
202020182
202020183
202020184
202020185
202020186
202020187
202020188
202020189
202020190
202020191
202020192
202020193
202020194
202020195
202020196
202020197
202020198
202020199
202020200
202020201
202020202
202020203
202020204
202020205
202020206
202020207
202020208
202020209
2020202010
2020202011
2020202012
2020202013
2020202014
2020202015
2020202016
2020202017
2020202018
2020202019
2020202020
2020202021
2020202022
2020202023
2020202024
2020202025
2020202026
2020202027
2020202028
2020202029
2020202030
2020202031
2020202032
2020202033
2020202034
2020202035
2020202036
2020202037
2020202038
2020202039
2020202040
2020202041
2020202042
2020202043
2020202044
2020202045
2020202046
2020202047
2020202048
2020202049
2020202050
2020202051
2020202052
2020202053
2020202054
2020202055
2020202056
2020202057
2020202058
2020202059
2020202060
2020202061
2020202062
2020202063
2020202064
2020202065
2020202066
2020202067
2020202068
2020202069
2020202070
2020202071
2020202072
2020202073
2020202074
2020202075
2020202076
2020202077
2020202078
2020202079
2020202080
2020202081
2020202082
2020202083
2020202084
2020202085
2020202086
2020202087
2020202088
2020202089
2020202090
2020202091
2020202092
2020202093
2020202094
2020202095
2020202096
2020202097
2020202098
2020202099
20202020100
20202020101
20202020102
20202020103
20202020104
20202020105
20202020106
20202020107
20202020108
20202020109
20202020110
20202020111
20202020112
20202020113
20202020114
20202020115
20202020116
20202020117
20202020118
20202020119
20202020120
20202020121
20202020122
20202020123
20202020124
20202020125
20202020126
20202020127
20202020128
20202020129
20202020130
20202020131
20202020132
20202020133
20202020134
20202020135
20202020136
20202020137
20202020138
20202020139
20202020140
20202020141
20202020142
20202020143
20202020144
20202020145
20202020146
20202020147
20202020148
20202020149
20202020150
20202020151
20202020152
20202020153
20202020154
20202020155
20202020156
20202020157
20202020158
20202020159
20202020160
20202020161
20202020162
20202020163
20202020164
20202020165
20202020166
20202020167
20202020168
20202020169
20202020170
20202020171
20202020172
20202020173
20202020174
20202020175
20202020176
20202020177
20202020178
20202020179
20202020180
20202020181
20202020182
20202020183
20202020184
20202020185
20202020186
20202020187
20202020188
20202020189
20202020190
20202020191
20202020192
20202020193
20202020194
20202020195
20202020196
20202020197
20202020198
20202020199
20202020200
20202020201
20202020202
20202020203
20202020204
20202020205
20202020206
20202020207
20202020208
20202020209
202020202010
202020202011
202020202012
202020202013
202020202014
202020202015
202020202016
202020202017
202020202018
202020202019
202020202020
202020202021
202020202022
202020202023
202020202024
202020202025
202020202026
202020202027
202020202028
202020202029
202020202030
202020202031
202020202032
202020202033
202020202034
202020202035
202020202036
202020202037
202020202038
202020202039
202020202040
202020202041
202020202042
202020202043
202020202044
202020202045
202020202046
202020202047
202020202048
202020202049
202020202050
202020202051
202020202052
202020202053
202020202054
202020202055
202020202056
202020202057
202020202058
202020202059
202020202060
202020202061
202020202062
202020202063
202020202064
202020202065
202020202066
202020202067
202020202068
202020202069
202020202070
202020202071
202020202072
202020202073
202020202074
202020202075
202020202076
202020202077
202020202078
202020202079
202020202080
202020202081
202020202082
202020202083
202020202084
202020202085
202020202086
202020202087
202020202088
202020202089
202020202090
202020202091
202020202092
202020202093
202020202094
202020202095
202020202096
202020202097
202020202098
202020202099
2020202020100
2020202020101
2020202020102
2020202020103
2020202020104
2020202020105
2020202020106
2020202020107
2020202020108
2020202020109
2020202020110
2020202020111
2020202020112
2020202020113
2020202020114
2020202020115
2020202020116
2020202020117
2020202020118
2020202020119
2020202020120
2020202020121
2020202020122
2020202020123
2020202020124
2020202020125
2020202020126
2020202020127
2020202020128
2020202020129
2020202020130
2020202020131
2020202020132
2020202020133
2020202020134
2020202020135
2020202020136
2020202020137
2020202020138
2020202020139
2020202020140
2020202020141
2020202020142
2020202020143
2020202020144
2020202020145
20
```

```

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

```

```

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

```

```

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

```

```

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

```

```

2375     fmt.Printf("\n")
2376   }
2377 }
2378
2379 // <a name="history">History</a>
2380 // these should be browsed and edited by HTTP browser
2381 // show the time of command with -t and direcotry with -ls
2382 // openfile-history, sort by -a -m -c
2383 // sort by elapsed time by -t -s
2384 // search by "more" like interface
2385 // edit history
2386 // sort history, and wc or uniq
2387 // CPU and other resource consumptions
2388 // limit showing range (by time or so)
2389 // export / import history
2390 func (gshCtx *GshContext)xHistory(argv []string){
2391   atWorkDirX := -1
2392   if 1 < len(argv) && strBegins(argv[1],"@") {
2393     atWorkDirX,_ = strconv.Atoi(argv[1][1:])
2394   }
2395   //fmt.Printf("--D-- showHistory@\n",argv)
2396   for i, v := range gshCtx.CommandHistory {
2397     // exclude commands not to be listed by default
2398     // internal commands may be suppressed by default
2399     if v.CmdLine == "" && !isin("-a",argv) {
2400       continue;
2401     }
2402     if 0 <= atWorkDirX {
2403       if v.WorkDirX != atWorkDirX {
2404         continue
2405       }
2406     if !isin("-n",argv){ // like "fc"
2407       fmt.Printf("!%-2d ",i)
2408     }
2409     if isin("-v",argv){
2410       fmt.Println(v) // should be with it date
2411     }else{
2412       if isin("-l",argv) || isin("-10",argv) {
2413         elps := v.EndAt.Sub(v.StartAt);
2414         start := v.StartAt.Format(time.Stamp)
2415         fmt.Printf("%d ",v.WorkDirX)
2416         fmt.Printf("[%v] %1v/t ",start,elps)
2417       }
2418       if isin("-1",argv) && !isin("-10",argv){
2419         fmt.Printf("%v",Rusagef("%t %1v/t %s",argv,v.Rusagev))
2420       }
2421       if isin("-at",argv) { // isin("-ls",argv){
2422         dhi := v.WorkDirX // workdir history index
2423         fmt.Printf("%d %s\t",dhi,v.WorkDir)
2424         // show the FileInfo of the output command??
2425       }
2426       fmt.Printf("%s",v.CmdLine)
2427       fmt.Printf("\n")
2428     }
2429   }
2430 }
2431 // !n - history index
2432 func searchHistory(gshCtx GshContext, gline string) (string, bool, bool){
2433   if gline[0] == '!' {
2434     hix, err := strconv.Atoi(gline[1:])
2435     if err != nil {
2436       fmt.Printf("--E-- (%s : range)\n",hix)
2437       return "", false, true
2438     }
2439     if hix < 0 || len(gshCtx.CommandHistory) <= hix {
2440       fmt.Printf("--E-- (%d : out of range)\n",hix)
2441       return "", false, true
2442     }
2443   }
2444   return gshCtx.CommandHistory[hix].CmdLine, false, false
2445 }
2446 // search
2447 //for i, v := range gshCtx.CommandHistory {
2448 //}
2449 return gline, false, false
2450 }
2451 func (gsh*GshContext)cmdStringInHistory(hix int)(cmd string, ok bool){
2452   if 0 <= hix && hix < len(gsh.CommandHistory) {
2453     return gsh.CommandHistory[hix].CmdLine,true
2454   }
2455   return "",false
2456 }
2457
2458 // temporary adding to PATH environment
2459 // cd name -lib for LD_LIBRARY_PATH
2460 // chdir with directory history (date + full-path)
2461 // -s for sort option (by visit date or so)
2462 func (gsh*GshContext>ShowChdirHistory(i int,v GChdirHistory, argv []string){
2463   fmt.Printf("!%-2d ",v.CmdIndex) // the first command at this WorkDir
2464   fmt.Printf("%d ",i)
2465   fmt.Printf("[%v] %v.Movedat.Format(time.Stamp)
2466   showFileInfo(v.Dir,argv)
2467 }
2468 func (gsh*GshContext>ShowChdirHistory(argv []string){
2469   for i, v := range gsh.ChdirHistory {
2470     gsh.ShowChdirHistory1(i,v,argv)
2471   }
2472 }
2473 func skipOpts(argv[]string)(int){
2474   for i,v := range argv {
2475     if strBegins(v,"-") {
2476     }else{
2477       return i
2478     }
2479   }
2480   return -1
2481 }
2482 func (gshCtx*GshContext)xChdir(argv []string){
2483   cdhist := gshCtx.ChdirHistory
2484   if isin("?",argv) || isin("-t",argv) || isin("-a",argv) {
2485     gshCtx.ShowChdirHistory(argv)
2486     return
2487   }
2488   pwd, _ := os.Getwd()
2489   dir := ""
2490   if len(argv) <= 1 {
2491     dir = toFullPath("~/")
2492   }else{
2493     i := skipOpts(argv[1:])
2494     if i < 0 {
2495       dir = toFullPath("~/")
2496     }else{
2497       dir = argv[1+i]
2498     }
2499 }

```

```

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

```

```

2625     showUsage(argv[1], argv, &gshCtx.LastRusage)
2626     rusagev := RusageSubv(rusagev2, rusagev1)
2627     showUsage("self", argv, &rusagev[0])
2628     showUsage("chld", argv, &rusagev[1])
2629     return fin
2630   }else{
2631     rusage:= syscall.Rusage {}
2632     syscall.Getrusage(syscall.RUSAGE_SELF,&rusage)
2633     showUsage("self",argv, &rusage)
2634     syscall.Getrusage(syscall.RUSAGE_CHILDREN,&rusage)
2635     showUsage("chld",argv, &rusage)
2636     return false
2637   }
2638 }
2639 func (gshCtx *GshContext)xJobs(argv[]string){
2640   fmt.Printf("--# Jobs\n",len(gshCtx.BackGroundJobs))
2641   for ji, pid := range gshCtx.BackGroundJobs {
2642     //wstat := syscall.WaitStatus {0}
2643     rusage := syscall.Rusage {}
2644     //wpid, err := syscall.Wait4(pid,&wstat,syscall.WNOHANG,&rusage);
2645     wpid, err := syscall.Wait4(pid,nil,syscall.WNOHANG,&rusage);
2646     if err != nil {
2647       fmt.Printf("--E-- %%d [%d] (%v)\n",ji, pid, err)
2648     }else{
2649       fmt.Printf("%%d[%d](%d)\n",ji, pid, wpid)
2650       showUsage("chld",argv,&rusage)
2651     }
2652   }
2653 }
2654 func (gsh*GshContext)inBackground(argv[]string)(bool){
2655   if gsh.CmdTrace { fmt.Printf("--I-- inBackground(%v)\n",argv) }
2656   gsh.BackGround = true // set background option
2657   xfin := false
2658   xfin = gsh.gshellv(argv)
2659   gsh.BackGround = false
2660   return xfin
2661 }
2662 // -o file without command means just opening it and refer by #N
2663 // should be listed by "files" command
2664 func (gshCtx*xGshContext)xOpen(argv[]string){
2665   var pv = []int{-1,-1}
2666   err := syscall.Pipe(pv)
2667   fmt.Printf("--I-- pipe()=[#%d,#%d](%v)\n",pv[0],pv[1],err)
2668 }
2669 func (gshCtx*GshContext)fromPipe(argv[]string){
2670 }
2671 func (gshCtx*xGshContext)xClose(argv[]string){
2672 }
2673
2674 // <a name="redirect">redirect</a>
2675 func (gshCtx*xGshContext)redirect(argv[]string)(bool){
2676   if len(argv) < 2 {
2677     return false
2678   }
2679
2680   cmd := argv[0]
2681   fname := argv[1]
2682   var file *os.File = nil
2683
2684   fdiix := 0
2685   mode := os.O_RDONLY
2686
2687   switch {
2688   case cmd == "-i" || cmd == "<":
2689     fdiix = 0
2690     mode = os.O_RDONLY
2691   case cmd == "-o" || cmd == ">":
2692     fdiix = 1
2693     mode = os.O_RDWR | os.O_CREATE
2694   case cmd == "-a" || cmd == ">>":
2695     fdiix = 1
2696     mode = os.O_RDWR | os.O_CREATE | os.O_APPEND
2697   }
2698   if fname[0] == '#' {
2699     fd, err := strconv.Atoi(fname[1:])
2700     if err != nil {
2701       fmt.Printf("--E-- (%v)\n",err)
2702       return false
2703     }
2704     file = os.NewFile(uintptr(fd),"MaybePipe")
2705   }else{
2706     xfile, err := os.OpenFile(argv[1], mode, 0600)
2707     if err != nil {
2708       fmt.Printf("--E-- (%s)\n",err)
2709       return false
2710     }
2711     file = xfile
2712   }
2713   gshPA := gshCtx.gshPA
2714   savfd := gshPA.Files[fdiix]
2715   gshPA.Files[fdiix] = file.Fd()
2716   fmt.Printf("--I-- Opened [%d] %s\n",file.Fd(),argv[1])
2717   gshCtx.gshellv(argv[2:])
2718   gshPA.Files[fdiix] = savfd
2719
2720   return false
2721 }
2722
2723 //fmt.Fprintf(res, "GShell Status: %q", html.EscapeString(req.URL.Path))
2724 func httpHandler(res http.ResponseWriter, req *http.Request){
2725   path := req.URL.Path
2726   fmt.Printf("--I-- Got HTTP Request(%s)\n",path)
2727   {
2728     gshCtxtBuf, _ := setupGshContext()
2729     gshCtx := &gshCtxtBuf
2730     fmt.Printf("--I-- %s\n",path[1:])
2731     gshCtx.tgshell(path[1:])
2732   }
2733   fmt.Fprintf(res, "Hello(^~^)/\n%s\n",path)
2734 }
2735 func (gshCtx *GshContext) httpServer(argv []string){
2736   http.HandleFunc("/", httpHandler)
2737   accport := "localhost:9999"
2738   fmt.Printf("--I-- HTTP Server Start at [%s]\n",accport)
2739   http.ListenAndServe(accport,nil)
2740 }
2741 func (gshCtx *GshContext)xGo(argv[]string){
2742   go gshCtx.gshellv(argv[1:]);
2743 }
2744 func (gshCtx *GshContext) xPs(argv[]string)(){
2745 }
2746
2747 // <a name="plugin">Plugin</a>
2748 // plugin [-ls [names]] to list plugins
2749 // Reference: <a href="https://golang.org/src/plugin/">plugin</a> source code

```

```

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

```

```

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

```

```

3000 func (gsh*GshContext)Deq(argv[]string){
3001 }
3002 func (gsh*GshContext)Push(argv[]string{
3003     gsh.iValStack = append(gsh.iValStack,argv[1:])
3004     fmt.Printf("depth=%d\n",len(gsh.iValStack))
3005 }
3006 func (gsh*GshContext)Dump(argv[]string){
3007     for i,v := range gsh.iValStack {
3008         fmt.Printf("%d %v\n",i,v)
3009     }
3010 }
3011 func (gsh*GshContext)Pop(argv[]string){
3012     depth := len(gsh.iValStack)
3013     if 0 < depth {
3014         v := gsh.iValStack[depth-1]
3015         if isin("-cat",argv){
3016             gsh.iValues = append(gsh.iValues,v...)
3017         }else{
3018             gsh.iValues = v
3019         }
3020         gsh.iValStack = gsh.iValStack[0:depth-1]
3021         fmt.Printf("depth=%d %s\n",len(gsh.iValStack),gsh.iValues)
3022     }else{
3023         fmt.Printf("depth=%d\n",depth)
3024     }
3025 }
3026
3027 // <a name="interpreter">Command Interpreter</a>
3028 func (gshCtx*GshContext)gshellv(argv []string) (fin bool) {
3029     fin = false
3030
3031     if gshCtx.CmdTrace { fmt.Fprintf(os.Stderr,"--I-- gshellv(%d)\n",len(argv)) }
3032     if len(argv) <= 0 {
3033         return false
3034     }
3035     xargv := []string{}
3036     for ai := 0; ai < len(argv); ai++ {
3037         xargv = append(xargv,strsubst(gshCtx,argv[ai],false))
3038     }
3039     argv = xargv
3040     if false {
3041         for ai := 0; ai < len(argv); ai++ {
3042             fmt.Printf("[%d] %s[%d]\n",
3043                         ai,argv[ai],len(argv[ai]),argv[ai])
3044         }
3045     }
3046     cmd := argv[0]
3047     if gshCtx.CmdTrace { fmt.Fprintf(os.Stderr,"--I-- gshellv(%d)%v\n",len(argv),argv) }
3048     switch { // https://tour.golang.org/flowcontrol/11
3049     case cmd == "":
3050         gshCtx.xPwd([]string{}) // emtpy command
3051     case cmd == "-x":
3052         gshCtx.CmdTrace = ! gshCtx.CmdTrace
3053     case cmd == "-xt":
3054         gshCtx.CmdTime = ! gshCtx.CmdTime
3055     case cmd == "-ot":
3056         gshCtx.sconnect(true, argv)
3057     case cmd == "-on":
3058         gshCtx.sconnect(false, argv)
3059     case cmd == "-it":
3060         gshCtx.saccept(true , argv)
3061     case cmd == "-in":
3062         gshCtx.saccept(false, argv)
3063     case cmd == "-i" || cmd == "<" || cmd == "-o" || cmd == ">" || cmd == "-a" || cmd == ">>" || cmd == "-s" || cmd == "><":
3064         gshCtx.redirect(argv)
3065     case cmd == "|":
3066         gshCtx.fromPipe(argv)
3067     case cmd == "args":
3068         gshCtx.Args(argv)
3069     case cmd == "bg" || cmd == "-bg":
3070         rfin := gshCtx.inBackground(argv[1:])
3071         return rfin
3072     case cmd == "-bn":
3073         gshCtx.Basename(argv)
3074     case cmd == "call":
3075         _ = gshCtx.excommand(false,argv[1:])
3076     case cmd == "cd" || cmd == "chdir":
3077         gshCtx.xChdir(argv);
3078     case cmd == "-cksum":
3079         gshCtx.xFind(argv)
3080     case cmd == "-sum":
3081         gshCtx.xFind(argv)
3082     case cmd == "close":
3083         gshCtx.xClose(argv)
3084     case cmd == "gcp":
3085         gshCtx.FileCopy(argv)
3086     case cmd == "dec" || cmd == "decode":
3087         gshCtx.Dec(argv)
3088     case cmd == "#define":
3089     case cmd == "dump":
3090         gshCtx.Dump(argv)
3091     case cmd == "echo":
3092         echo(argv,true)
3093     case cmd == "enc" || cmd == "encode":
3094         gshCtx.Enc(argv)
3095     case cmd == "env":
3096         env(argv)
3097     case cmd == "eval":
3098         xEval(argv[1:],true)
3099     case cmd == "exec":
3100         _ = gshCtx.excommand(true,argv[1:])
3101         /* should not return here
3102     case cmd == "exit" || cmd == "quit":
3103         // write Result code EXIT to 3>
3104         return true
3105     case cmd == "fds":
3106         // dump the attributes of fds (of other process)
3107     case cmd == "find" || cmd == "fin" || cmd == "ufind" || cmd == "uf":
3108         gshCtx.xFind(argv[1:])
3109     case cmd == "fu":
3110         gshCtx.xFind(argv[1:])
3111     case cmd == "fork":
3112         // mainly for a server
3113     case cmd == "-gen":
3114         gshCtx.gen(argv)
3115     case cmd == "-go":
3116         gshCtx.xGo(argv)
3117     case cmd == "-grep":
3118         gshCtx.xFind(argv)
3119     case cmd == "gdeq":
3120         gshCtx.Deq(argv)
3121     case cmd == "geng":
3122         gshCtx.Eng(argv)
3123     case cmd == "gpop":
3124         gshCtx.Pop(argv)

```

```

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

```

```

3250 }
3251 func (gshCtx *GshContext) ttyline()(*os.File){
3252     file, err := os.OpenFile(gshCtx.ttyfile(),os.O_RDWR|os.O_CREATE|os.O_TRUNC,0600)
3253     if err != nil {
3254         fmt.Printf("--F-- cannot open %s (%s)\n",gshCtx.ttyfile(),err)
3255         return file;
3256     }
3257     return file
3258 }
3259 func (gshCtx *GshContext)getline(hix int, skipping bool, prevline string) (string) {
3260     if( skipping ){
3261         reader := bufio.NewReaderSize(os.Stdin,LINESIZE)
3262         line, _, _:= reader.ReadLine()
3263         return string(line)
3264     }else
3265     if true {
3266         return xgetline(hix,prevline,gshCtx)
3267     }
3268 /*
3269 else
3270 if( with_xgetline && gshCtx.GetLine != "" ){
3271     //var xhix int64 = int64(hix); // cast
3272     newenv := os.Environ()
3273     newenv = append(newenv, "GSH_LINENO="+strconv.FormatInt(int64(hix),10) )
3274
3275     tty := gshCtx.ttyline()
3276     tty.WriteString(prevline)
3277     Pa := os.ProcAttr {
3278         "", // start dir
3279         newenv, //os.Environ(),
3280         []*os.File{os.Stdin,os.Stdout,os.Stderr,tty},
3281         nil,
3282     }
3283 //fmt.Printf("--I-- getline=%s // %s\n",gsh_getlinev[0],gshCtx.GetLine)
3284 proc, err := os.StartProcess(gsh_getlinev[0],[]string{"getline","getline"},&Pa)
3285     if err != nil {
3286         fmt.Printf("--F-- getline process error (%v)\n",err)
3287         // for ; ; {}
3288         return "exit (getline program failed)"
3289     }
3290     //stat, err := proc.Wait()
3291     proc.Wait()
3292     buff := make([]byte,LINESIZE)
3293     count, err := tty.Read(buff)
3294     //_, err = tty.Read(buff)
3295     //fmt.Printf("--D-- getline (%d)\n",count)
3296     if err != nil {
3297         if ! (count == 0) { // && err.String() == "EOF" ) {
3298             fmt.Printf("--E-- getline error (%s)\n",err)
3299         }
3300     }else{
3301         //fmt.Printf("--I-- getline OK \"%s\"\n",buff)
3302     }
3303     tty.Close()
3304     gline := string(buff[0:count])
3305     return gline
3306 }
3307 */
3308 {
3309     // if isatty {
3310     fmt.Printf("!%d",hix)
3311     fmt.Print(PROMPT)
3312     //}
3313     reader := bufio.NewReaderSize(os.Stdin,LINESIZE)
3314     line, _, _:= reader.ReadLine()
3315     return string(line)
3316 }
3317 }
3318 //== begin ===== getline =====
3319 /*
3320 * getline.c
3321 * 2020-0819 extracted from dog.c
3322 * getline.go
3323 * 2020-0822 ported to Go
3324 */
3325 /*
3326 */
3327 package main // getline main
3328 import (
3329     "fmt"      // <a href="https://golang.org/pkg/fmt/">fmt</a>
3330     "strings"   // <a href="https://golang.org/pkg/strings/">strings</a>
3331     "os"        // <a href="https://golang.org/pkg/os/">os</a>
3332     "syscall"   // <a href="https://golang.org/pkg/syscall/">syscall</a>
3333     //"bytes"    // <a href="https://golang.org/pkg/os/">os</a>
3334     //"os/exec"  // <a href="https://golang.org/pkg/os/">os</a>
3335 )
3336 */
3337
3338 // C language compatibility functions
3339 var errno = 0
3340 var stdin *os.File = os.Stdin
3341 var stdout *os.File = os.Stdout
3342 var stderr *os.File = os.Stderr
3343 var EOF = -1
3344 var NULL = 0
3345 type FILE os.File
3346 type StrBuff []byte
3347 var NULL_FPC *os.File = nil
3348 var NULLSP = 0
3349 //var LINESIZE = 1024
3350
3351 func system(cmdstr string)(int{
3352     PA := syscall.ProcAttr {
3353         "", // the starting directory
3354         os.Environ(),
3355         [uintptr(os.Stdin.Fd()),os.Stdout.Fd(),os.Stderr.Fd()],
3356         nil,
3357     }
3358     argv := strings.Split(cmdstr," ")
3359     pid,err := syscall.ForkExec(argv[0],argv,&PA)
3360     if( err != nil ){
3361         fmt.Printf("--E-- syscall(%v) err(%v)\n",cmdstr,err)
3362     }
3363     syscall.Wait4(pid,nil,0,nil)
3364
3365 /*
3366 argv := strings.Split(cmdstr," ")
3367 fmt.Fprintf(os.Stderr,"--I-- system(%v)\n",argv)
3368 //cmd := exec.Command(argv[0]...)
3369 cmd := exec.Command(argv[0],argv[1],argv[2])
3370 cmd.Stdin = strings.NewReader("output of system")
3371 var out bytes.Buffer
3372 cmd.Stdout = &out
3373 var serr bytes.Buffer
3374 cmd.Stderr = &serr

```

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

```

```

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

```

```

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

```

```

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

```

```

3875     case 'L'-0x40:
3876         redraw(lno,line,right);
3877         continue;
3878     case 'K'-0x40:
3879         right = "";
3880         redraw(lno,line,right);
3881         continue;
3882     case 'E'-0x40:
3883         line += right
3884         right = ""
3885         redraw(lno,line,right);
3886         continue;
3887     case 'A'-0x40:
3888         right = line + right
3889         line = ""
3890         redraw(lno,line,right);
3891         continue;
3892     case 'U'-0x40:
3893         line = ""
3894         right = ""
3895         clearline();
3896         redraw(lno,line,right);
3897         continue;
3898     case 0x7f: // DEL
3899         if( 0 < len(line) ){
3900             line,_ = delTailChar(line)
3901             redraw(lno,line,right);
3902         }
3903         continue;
3904     case 'H'-0x40:
3905         if( 0 < len(line) ){
3906             line,_ = delTailChar(line)
3907             redraw(lno,line,right);
3908         }
3909         continue;
3910     if( ch == '\n' || ch == '\r' ){
3911         fputc(ch,stderr);
3912         break;
3913     }
3914     line += string(ch);
3915     redraw(lno,line,right);
3916 }
3917 EXIT:
3918 system("/bin/stty echo sane");
3920 }
3921 //fprintf(stderr,"%r\nLINE:%s\r\n",line);
3922
3923 EXIT_GOT:
3924     return line + right;
3925 }
3926
3927 func getline_main(){
3928     line := xgetline(0,"",nil)
3929     fprintf(stderr,"%s\n",line);
3930 /* */
3931     dp = strpbrk(line,"\r\n");
3932     if( dp != NULL ){
3933         *dp = 0;
3934     }
3935
3936     if( 0 ){
3937         fprintf(stderr,"%n(%d)\n",int(strlen(line)));
3938     }
3939     if( lseek(3,0,0) == 0 ){
3940         if( romkanemode ){
3941             var buf [8*1024]byte;
3942             convs(line,buf);
3943             strcpy(line,buf);
3944         }
3945         write(3,line,strlen(line));
3946         ftruncate(3,lseek(3,0,SEEK_CUR));
3947         //fprintf(stderr,"outsize=%d\n",(int)lseek(3,0,SEEK_END));
3948         lseek(3,0,SEEK_SET);
3949         close(3);
3950     }else{
3951         fprintf(stderr,"%r\ngetline: ");
3952         trans(line);
3953         //printf("%s\n",line);
3954         printf("\n");
3955     }
3956 */
3957 }
3958 //== end ===== getline
3959
3960 //
3961 // $USERHOME/.gsh/
3962 //     gsh-rc.txt, or gsh-configure.txt
3963 //     gsh-history.txt
3964 //     gsh-aliases.txt // should be conditional?
3965 //
3966 func (gshCtx *GshContext)gshSetupHomedir()(bool) {
3967     homedir,found := userHomeDir()
3968     if !found {
3969         fmt.Println("--E-- You have no UserHomeDir\n")
3970         return true
3971     }
3972     gshhome := homedir + "/" + GSH_HOME
3973     _,err2 := os.Stat(gshhome)
3974     if err2 != nil {
3975         err3 := os.Mkdir(gshhome,0700)
3976         if err3 != nil {
3977             fmt.Println("--E-- Could not Create %s (%s)\n",
3978                     gshhome,err3)
3979             return true
3980         }
3981         fmt.Println("--I-- Created %s\n",gshhome)
3982     }
3983     gshCtx.GshHomeDir = gshhome
3984     return false
3985 }
3986 func setupGshContext()(GshContext,bool){
3987     gshPA := syscall.ProcAttr {
3988         "", // the staring directory
3989         os.Environ(), // environ[]
3990         []uintptr{os.Stdin.Fd(),os.Stdout.Fd(),os.Stderr.Fd()},
3991         nil, // OS specific
3992     }
3993     cwd, _ := os.Getwd()
3994     gshctx := GshContext {
3995         cwd, // StartDir
3996         "", // GetLine
3997         []GChdirHistory { { cwd,time.Now(),0 } }, // ChdirHistory
3998         gshPA,
3999         []GCommandHistory{}, //something for invocation?

```

```

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

```

```

4125     hix++;
4126   }
4127   return *gshCtx
4128 }
4129 func main() {
4130   gshCtxBuf := GshContext{}
4131   gsh := &gshCtxBuf
4132   argv := os.Args
4133   if 1 < len(argv) {
4134     if isin("version", argv) {
4135       gsh.showVersion(argv)
4136       return
4137     }
4138     comx := isinX("-c", argv)
4139     if 0 < comx {
4140       gshCtxBuf,err := setupGshContext()
4141       gsh := &gshCtxBuf
4142       if !err {
4143         gsh.gshellv(argv[comx+1:])
4144       }
4145     }
4146   }
4147   if 1 < len(argv) && isin("-s", argv) {
4148 }else{
4149   gsh.showVersion(append(argv,[]string{"-l","-a"}...))
4150 }
4151 script(nil)
4152 //gshCtx := script(nil)
4153 //gshell(gshCtx,"time")
4154 }
4155 //</div></details>
4156 //<details id="todo"><summary>Consideration</summary><div class="gsh-src">
4157 // - inter gsh communication, possibly running in remote hosts -- to be remote shell
4158 // - merged histories of multiple parallel gsh sessions
4159 // - alias as a function or macro
4160 // - instant alias and environ export to the permanent > ~/.gsh/gsh-alias and gsh-environ
4161 // - retrieval PATH of files by its type
4162 // - gsh as an IME with completion using history and file names as dictionaires
4163 // - gsh a scheduler in precise time of within a millisecond
4164 // - all commands have its subcommand after "___" symbol
4165 // - filename expansion by "-find" command
4166 // - history of ext code and output of each command
4167 // - "script" output for each command by pty-tee or telnet-tee
4168 // - $BUILTIN command in PATH to show the priority
4169 // - "?" symbol in the command (not as in arguments) shows help request
4170 // - searching command with wild card like: which ssh-*
4171 // - customizing by building plugin and dynamically linking it
4172 // - generating syntactic element like "if" by macro expansion (like CPP) >> alias
4173 // - "!" symbol should be used for negation, don't wast it just for job control
4174 // - don't put too long output to tty, record it into GSH_HOME/session-id/comand-id.log
4175 // - making canonical form of command at the start adding quatation or white spaces
4176 // - name(a,b,c) ... use "(" and ")" to show both delimiter and realm
4177 // - name? or name! might be useful
4178 // - tar format - packing directory contents into a single html file using data scheme
4179 // - filepath substitution shold be done by each command, especially in case of builins
4180 // - @N substitution for the history of working directory, and @spec for more generic ones
4181 // - @dir prefix to do the command at there, that means like (chdir @dir; command)
4182 // - GSH_PATH for plugins
4183 // - standard command output: list of data with name, size, resouce usage, modified time
4184 // - generic sort key option -nm name, -sz size, -ru rusage, -ts start-time, -tm mod-time
4185 // - wc word-count, grep match line count, ...
4186 // - -taifl-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 queier name)
4189 // - IME "?" at the top of the command line means searching history
4190 // - IME id@0x10000/ *xffff/
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 <div id="gsh-footer" style="">Fin.</div>
4212 <style id="gsh-style">
4213 #gsh {border-width:1px; margin:0; padding:0;}
4214 #gsh {font-family:monospace,Courier New; color:#ddf; font-size:8px;}
4215 #gsh{height:100px;}
4216 #xgsh header{height:100px; background-image:url(GShell-Logo00.png);}
4217 #gsh-menu{font-size:14pt; color:#f88;}
4218 #gsh-footer{height:100px; background-size:80px; background-repeat:no-repeat;}
4219 #gsh note{color:#000; font-size:10pt;}
4220 #gsh h2{color:#24a; font-family:Georgia; font-size:18pt;}
4221 #gsh details{color:#888; background-color:#aaa; font-family:monospace;}
4222 #gsh summary{font-size:16pt; color:#24a; background-color:#eef; height:30px;}
4223 #gsh pre{font-size:11pt; color:#223; background-color:#faffff;}
4224 #gsh a{color:#24a;}
4225 #gsh a[name]{color:#24a; font-size:16pt;}
4226 #gsh .gsh-src{white-space:pre; font-family:monospace,Courier New; font-size:11pt;}
4227 #gsh .gsh-src{background-color:#faffff; color:#223;}
4228 #gsh-src-src{spellcheck:false}
4229 #src-frame-textarea{white-space:pre; font-family:monospace,Courier New; font-size:11pt;}
4230 #src-frame-textarea{background-color:#faffff; color:#223;}
4231 @media print {
4232   #gsh pre{font-size:11pt !important;}
4233 }
4234 </style>
4235 <!--
4236 // Logo image should be drawn by JavaScript from a meta-font.
4237 // CSS seems not follow line-splitited URL
4238 -->
4239 <script id="gsh-run">
4240 GshLogo="data:image/png;base64,%
4241 iVBORw0KGgoAAAANSUhEUgAAQEEAAB/CAYAAADvs3f4AAAAAXNSR0IArs4c6QAAAHHlWElm%
4242 TU0AKgAAAAGABAEaAUAAAABAAAABgPg5bAAAUAABAAAARgEoAAMAAAABAAIAIdpAAQAAAAB%
4243 AAAATgAAAAAAAABIAAAAQAEEAgAAAABAAQgAQADAAAQAQBAACAgQgAEAAAAAAQgQgAwAB%
4244 AAAAQAAAH8AAAAAYX1wsFlzAALEWAACxMBAJgCgAAAF3RJREFUEAHnQuUFNwZ%
4245 x++tUkZ31CggO/jY6OsB8wgMzAvn7uG4+b1STR7yNQxdQPCKGj2auWld2MsLRkeUaPnoCdu%
4246 4iuJx7jriYz50DGMF2VqIBEEisSggCoIMMA+m+u+//ZMD9Ulda6u2auBv91GKrq3Vvdx6/q%
4247

```



```

4375 function shiftBG(){
4376   bannerStop = !bannerStop
4377   document.getElementById('banner').style.backgroundPosition = "0 0";
4378 }
4379
4380 function html_fold(){
4381   document.getElementById('index').open=false
4382   document.getElementById('gsh-gocode').open=false
4383   document.getElementById('todo').open=false
4384   document.getElementById('reference').open=false
4385 }
4386 function html_open(){
4387   document.getElementById('index').open=true
4388   document.getElementById('gsh-gocode').open=true
4389   document.getElementById('todo').open=true
4390   document.getElementById('reference').open=true
4391 }
4392 function html_stop(){
4393   bannerStop = !bannerStop
4394 }
4395
4396 //https://www.w3schools.com/jsref/met_win_setinterval.asp
4397 function shiftBanner(){
4398   var now = new Date().getTime();
4399   //console.log("now="+now%10)
4400   if( !bannerStop ){
4401     document.getElementById('banner').style.backgroundPosition = ((now/10)%100000)+" 0";
4402   }
4403 }
4404 setInterval(shiftBanner,10);
4405
4406 // from embedded html to standalone page
4407 function html_close(){
4408   window.close()
4409 }
4410
4411 // from embedded html to standalone page
4412 function html_new(){
4413   newwin = window.open("", "", "");
4414   src = document.getElementById("gsh");
4415   newwin.document.write("/**"+html+"\n");
4416   newwin.document.write("<"+span id='gsh'>");
4417   newwin.document.write(src.innerHTML);
4418   newwin.document.write("<"+/span><"+/html>\n"); // gsh span
4419   newwin.document.close();
4420   newwin.focus();
4421 }
4422
4423 // source code view
4424 function frame_close(){
4425   srcframe = document.getElementById("src-frame");
4426   srcframe.innertHTML = "";
4427   //srcframe.style.cols = 1;
4428   srcframe.style.rows = 1;
4429   srcframe.style.height = 0;
4430   srcframe.style.display = false;
4431   src = document.getElementById("src-frame-textarea");
4432   src.innerHTML = ""
4433   //src.cols = 0
4434   src.rows = 0
4435   src.display = false
4436   //alert("--closed--")
4437 }
4438 //!-- | <span onclick="html_view();">Source</span> -->
4439 //| <span onclick="frame_close();">SourceClose</span> -->
4440 //| <span>Download</span> -->
4441 function frame_open(){
4442   oldsrc = document.getElementById("GENSRC");
4443   if( oldsrc != null ){
4444     //alert("--I--(erasing old text)")
4445     oldsrc.innertHTML = "";
4446     return
4447   }else{
4448     //alert("--I--(no old text)")
4449   }
4450   banner = document.getElementById('banner').style.backgroundImage;
4451   footer = document.getElementById('gsh-footer').style.backgroundImage;
4452   document.getElementById('banner').style.backgroundImage = "";
4453   document.getElementById('banner').style.backgroundPosition = "";
4454   document.getElementById('gsh-footer').style.backgroundImage = "";
4455
4456   src = document.getElementById("gsh");
4457   srcframe = document.getElementById("src-frame");
4458   srcframe.innertHTML = ""
4459   + "<"+cite id='GENSRC'>\n"
4460   + "<"+style>\n"
4461   + "#GENSRC textarea{tab-size:4;}\n"
4462   + "#GENSRC textarea{o-tab-size:4;}\n"
4463   + "#GENSRC textarea{moz-tab-size:4;}\n"
4464   + "#GENSRC textarea{spellcheck:false;}\n"
4465   + "<"+style>\n"
4466   + "<h2>\n"
4467   //+ "<"+span onclick='frame_close();'>Close</"+"span>\n"
4468   //+ " | <"+span onclick='html_stop();'>Run</"+"span>\n"
4469   + "</h2>\n"
4470   + "<"+textarea id='src-frame-textarea' cols=100 rows=40>"
4471   + "/<"+/html>\n"
4472   + "<"+span id='gsh'>"
4473   + src.innerHTML
4474   + "<"+/span><"+/html>\n"
4475   + "<"+/textarea>\n"
4476   + "<"/"+cite><!-- GENSRC -->\n";
4477
4478 //srcframe.style.cols = 80;
4479 //srcframe.style.rows = 80;
4480
4481 document.getElementById('banner').style.backgroundImage = banner;
4482 document.getElementById('gsh-footer').style.backgroundImage = footer
4483 }
4484 function html_view(){
4485   html_stop();
4486
4487   banner = document.getElementById('banner').style.backgroundImage;
4488   footer = document.getElementById('gsh-footer').style.backgroundImage;
4489   document.getElementById('banner').style.backgroundImage = "";
4490   document.getElementById('banner').style.backgroundPosition = "";
4491   document.getElementById('gsh-footer').style.backgroundImage = "";
4492
4493 //srcwin = window.open("", "CodeView2", "");
4494 //srcwin = window.open("", "", "");
4495 //srcwin.document.write("span id='gsh'>\n");
4496
4497 src = document.getElementById("gsh");
4498 srcwin.document.write("<style>\n");
4499 srcwin.document.write("textarea{tab-size:4;}\n");

```

```
4500 srcwin.document.write("textarearea{-o-tab-size:4;}\n");
4501 srcwin.document.write("textarearea{-moz-tab-size:4;}\n");
4502 srcwin.document.write("</style>\n");
4503 srcwin.document.write("<h2>\n");
4504 srcwin.document.write("<"+span onclick=\\"window.close();\\>Close</span> | \n");
4505 //srcwin.document.write("<"+span onclick=\\"html_stop();\\>Run</span>\n");
4506 srcwin.document.write("</h2>\n");
4507 srcwin.document.write("<textarea id=\\"gsh-src-src\\\" cols=100 rows=60>\n");
4508 srcwin.document.write("/*<"+html>\n");
4509 srcwin.document.write("<"+span id=\\"gsh\\>");
4510 srcwin.document.write(src.innerHTML);
4511 srcwin.document.write("<"+/span><"+/html>\n");
4512 srcwin.document.write("</"+"textarea>\n");
4513
4514 document.getElementById('banner').style.backgroundImage = banner;
4515 document.getElementById('gsh-footer').style.backgroundImage = footer
4516
4517 sty = document.getElementById("gsh-style");
4518 srcwin.document.write("<"+style>\n");
4519 srcwin.document.write(sty.innerHTML);
4520 srcwin.document.write("<"+/style>\n");
4521
4522 run = document.getElementById("gsh-run");
4523 srcwin.document.write("<"+script>\n");
4524 srcwin.document.write(run.innerHTML);
4525 srcwin.document.write("<"+/script>\n");
4526
4527 srcwin.document.write("<"+/span><"+/html>\n"); // gsh span
4528 srcwin.document.close();
4529 srcwin.focus();
4530 }
4531 </script>
4532 -->
4533 */ //</span></html>
4534
```