```
  1  /*<html>
  2  <span id="gsh">
  3  <meta charset="UTF-8">
  4  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  5  <link rel="icon" id="gsh-faviconurl" href=""><!-- place holder -->
  6  <span id="gsh-version" style="display:none;">gsh--0.2.9--2020-09-02--SatoxITS</span>
  7  <title>GShell-0.2.9 by SatoxITS</title>
  8  <header id="gsh-banner" height="100px" onclick="shiftBG();" style="">
  9  <div align="right"><note>GShell version 0.2.9 // 2020-09-02 // SatoxITS</note></div>
 10  </header>
 11  <h2>GShell // a General purpose Shell built on the top of Golang</h2>
 12  <p>
 13  <note>
 14  It is a shell for myself, by myself, of myself. --SatoxITS(^-^)
 15  </note>
 16  </p>
 17  <span id="gsh-WinId" onclick="win_jump('0.1');">0</span>
 18  <span id="gsh-menu">
 19  | <span id="gsh-menu-exit" onclick="html_close();"></span>
 20  | <span id="gsh-menu-fork" onclick="html_fork();">Fork</span>
 21  | <span id="gsh-menu-stop" onclick="html_stop(this,true);">Stop</span>
 22  | <span id="gsh-menu-fold" onclick="html_fold(this);">Unfold</span>
 23  | <span id="gsh-menu-cksum" onclick="html_cksum(this);">Cksum</span>
 24  <!-- | <span id="gsh-menu-pure" onclick="html_pure(this);">Pure</span> -->
 25  |</span>
 26  */
 27  /*
 28  <details id="gsh-statement" class="gsh-document"><summary>Statement</summary>
 29  <h3>Fun to create a shell</h3>
 30  <p>For a programmer, it must be far easy and fun to create his own simple shell
 31  rightly fitting to his favor and necessities, than learning existing shells with
 32  complex full features that he never use.
 33  I, as one of programmers, am writing this tiny shell for my own real needs,
 34  totally from scratch, with fun.
 35  </p><p>
 36  For a programmer, it is fun to learn new computer languages.  For long years before
 37  writing this software, I had been specialized to C and early HTML2 :-).
 38  Now writing this software,  I'm learning Go language, HTML5, JavaScript and CSS
 39  on demand as a novice of these, with fun.
 40  </p><p>
 41  This single file "gsh.go", that is executable by Go, contains all of the code written
 42  in Go. Also it can be displayed as "gsh.go.html" by browsers. It is a standalone
 43  HTML file that works as the viewer of the code of itself, and as the "home page" of
 44  this software.
 45  </p><p>
 46  Because this HTML file is a Go program, you may run it as a real shell program
 47  on your computer.
 48  But you must be aware that this program is written under situation like above.
 49  Needless to say, there is no warranty for this program in any means.
 50  </p>
 51  <address>Aug 2020, SatoxITS (sato@its-more.jp)</address>
 52  </details>
 53  */
 54  /*
 55  <details id="gsh-features" class="gsh-document"><summary>Features</summary><p>
 56  </p>
 57  <h3>Vi compatible command line editor</h3>
 58  <p>
 59  The command line of GShell can be edited with commands compatible with
 60  <a href="https://www.washington.edu/computing/unix/vi.html"><b>vi</b></a>.
 61  As in vi, you can enter <i><b>command mode</b></i> by <b>ESC</b> key,
 62  then move around in the history by <b><code>j k / ? n N</code></b>,
 63  or within the current line by <b><code>l h f w b 0 $ %</code></b> or so.
 64  </p>
 65  </details>
 66  */
 67  /*
 68  <details id="gsh-gindex">
 69  <summary>Index</summary><div class="gsh-src">
 70  Documents
 71      <span class="gsh-link" onclick="jumpto_JavaScriptView();">Command summary</span>
 72  Go lang part<span class="gsh-src" onclick="document.getElementById('gsh-gocode').open=true;">
 73      Package structures
 74          <a href="#import">import</a>
 75          <a href="#struct">struct</a>
 76      Main functions
 77          <a href="#comexpansion">str-expansion</a>   // macro processor
 78          <a href="#finder">finder</a>         // builtin find + du
 79          <a href="#grep">grep</a>         // builtin grep + wc + cksum + ...
 80          <a href="#plugin">plugin</a>        // plugin commands
 81          <a href="#ex-commands">system</a>       // external commands
 82          <a href="#builtin">builtin</a>      // builtin commands
 83          <a href="#network">network</a>      // socket handler
 84          <a href="#remote-sh">remote-sh</a>  // remote shell
 85          <a href="#redirect">redirect</a>     // StdIn/Out redireciton
 86          <a href="#history">history</a>      // command history
 87          <a href="#rusage">rusage</a>       // resouce usage
 88          <a href="#encode">encode</a>       // encode / decode
 89          <a href="#IME">IME</a>        // command line IME
 90          <a href="#getline">getline</a>      // line editor
 91          <a href="#scanf">scanf</a>       // string decomposer
 92          <a href="#interpreter">interpreter</a>  // command interpreter
 93          <a href="#main">main</a>
 94  </span>
 95  JavaScript part
 96      <a href="#script-src-view" class="gsh-link" onclick="jumpto_JavaScriptView();">Source</a>
 97      <a href="#gsh-data-frame" class="gsh-link" onclick="jumpto_DataView();">Builtin data</a>
 98  CSS part
 99      <a href="#style-src-view" class="gsh-link" onclick="jumpto_StyleView();">Source</a>
100  References
101      <a href="#" class="gsh-link" onclick="jumpto_WholeView();">Internal</a>
102      <a href="#gsh-reference" class="gsh-link" onclick="jumpto_ReferenceView();">External</a>
103  Whole parts
104      <a href="#whole-src-view" class="gsh-link" onclick="jumpto_WholeView();">Source</a>
105      <a href="#whole-src-view" class="gsh-link" onclick="jumpto_WholeView();">Download</a>
106      <a href="#whole-src-view" class="gsh-link" onclick="jumpto_WholeView();">Dump</a>
107
108  </div>
109  </details>
110  */
111  //<details id="gsh-gocode">
112  //<summary>Go Source</summary><div class="gsh-src" onclick="document.getElementById('gsh-gocode').open=false;">
113  // gsh - Go lang based Shell
114  // (c) 2020 ITS more Co., Ltd.
115  // 2020-0807 created by SatoxITS (sato@its-more.jp)
116
117  package main // gsh main
118  // <a name="import">Imported packages</a> // <a href="https://golang.org/pkg/">Packages</a>
119  import (
120      "fmt"        // <a href="https://golang.org/pkg/fmt/">fmt</a>
121      "strings"    // <a href="https://golang.org/pkg/strings/">strings</a>
122      "strconv"    // <a href="https://golang.org/pkg/strconv/">strconv</a>
123      "sort"       // <a href="https://golang.org/pkg/sort/">sort</a>
124      "time"       // <a href="https://golang.org/pkg/time/">time</a>
```

```
125        "bufio"      // <a href="https://golang.org/pkg/bufio/">bufio</a>
126        "io/ioutil" // <a href="https://golang.org/pkg/io/ioutil/">ioutil</a>
127        "os"         // <a href="https://golang.org/pkg/os/">os</a>
128        "syscall"    // <a href="https://golang.org/pkg/syscall/">syscall</a>
129        "plugin"     // <a href="https://golang.org/pkg/plugin/">plugin</a>
130        "net"        // <a href="https://golang.org/pkg/net/">net</a>
131        "net/http"   // <a href="https://golang.org/pkg/net/http/">http</a>
132        //"html"      // <a href="https://golang.org/pkg/html/">html</a>
133        "path/filepath" // <a href="https://golang.org/pkg/path/filepath/">filepath</a>
134        "go/types"   // <a href="https://golang.org/pkg/go/types/">types</a>
135        "go/token"   // <a href="https://golang.org/pkg/go/token/">token</a>
136        "encoding/base64"    // <a href="https://golang.org/pkg/encoding/base64/">base64</a>
137        "unicode/utf8"   // <a href="https://golang.org/pkg/unicode/utf8/">utf8</a>
138        //"gshdata" // gshell's logo and source code
139        "hash/crc32"      // <a href="https://golang.org/pkg/unicode/hash/crc32/">crc32</a>
140 )
141 const (
142        NAME = "gsh"
143        VERSION = "0.2.9"
144        DATE = "2020-09-02"
145        AUTHOR = "SatoxITS(^-^)/"
146 )
147 var (
148        GSH_HOME = ".gsh"   // under home directory
149        GSH_PORT = 9999
150        MaxStreamSize = int64(128*1024*1024) // 128GiB is too large?
151        PROMPT = "> "
152        LINESIZE = (8*1024)
153        PATHSEP = ":"   // should be ";" in Windows
154        DIRSEP = "/"    // canbe \ in Windows
155 )
156
157 // -xX logging control
158 // --A-- all
159 // --I-- info.
160 // --D-- debug
161 // --T-- time and resource usage
162 // --W-- warning
163 // --E-- error
164 // --F-- fatal error
165 // --Xn- network
166
167 // <a name="struct">Structures</a>
168 type GCommandHistory struct {
169        StartAt    time.Time // command line execution started at
170        EndAt      time.Time // command line execution ended at
171        ResCode    int       // exit code of (external command)
172        CmdError   error     // error string
173        OutData    *os.File  // output of the command
174        FoundFile  []string  // output - result of ufind
175        Rusagev    [2]syscall.Rusage // Resource consumption, CPU time or so
176        CmdId      int       // maybe with identified with arguments or impact
177                            // redireciton commands should not be the CmdId
178        WorkDir    string    // working directory at start
179        WorkDirX   int       // index in ChdirHistory
180        CmdLine    string    // command line
181 }
182 type GChdirHistory struct {
183        Dir     string
184        MovedAt    time.Time
185        CmdIndex   int
186 }
187 type CmdMode struct {
188        BackGround  bool
189 }
190 type Event struct {
191        when        time.Time
192        event       int
193        evarg       int64
194        CmdIndex    int
195 }
196 var CmdIndex int
197 var Events []Event
198 type PluginInfo struct {
199        Spec      *plugin.Plugin
200        Addr      plugin.Symbol
201        Name      string // maybe relative
202        Path      string // this is in Plugin but hidden
203 }
204 type GServer struct {
205        host       string
206        port       string
207 }
208
209 // <a href="https://tools.ietf.org/html/rfc3230">Digest</a>
210 const ( // SumType
211        SUM_ITEMS    = 0x000001 // items count
212        SUM_SIZE     = 0x000002 // data length (simplly added)
213        SUM_SIZEHASH   = 0x000004 // data length (hashed sequence)
214        SUM_DATEHASH   = 0x000008 // date of data (hashed sequence)
215        // also envelope attributes like time stamp can be a part of digest
216        // hashed value of sizes or mod-date of files will be useful to detect changes
217
218        SUM_WORDS    = 0x000010 // word count is a kind of digest
219        SUM_LINES    = 0x000020 // line count is a kind of digest
220        SUM_SUM64    = 0x000040 // simple add of bytes, useful for human too
221
222        SUM_SUM32_BITS   = 0x000100 // the number of true bits
223        SUM_SUM32_2BYTE = 0x000200 // 16bits words
224        SUM_SUM32_4BYTE = 0x000400 // 32bits words
225        SUM_SUM32_8BYTE = 0x000800 // 64bits words
226
227        SUM_SUM16_BSD   = 0x001000 // UNIXsum -sum -bsd
228        SUM_SUM16_SYSV  = 0x002000 // UNIXsum -sum -sysv
229        SUM_UNIXFILE    = 0x004000
230        SUM_CRCIEEE = 0x008000
231 )
232 type CheckSum struct {
233        Files      int64   // the number of files (or data)
234        Size       int64   // content size
235        Words      int64   // word count
236        Lines      int64   // line count
237        SumType    int
238        Sum64      uint64
239        Crc32Table  crc32.Table
240        Crc32Val   uint32
241        Sum16      int
242        Ctime      time.Time
243        Atime      time.Time
244        Mtime      time.Time
245        Start      time.Time
246        Done       time.Time
247        RusgAtStart [2]syscall.Rusage
248        RusgAtEnd   [2]syscall.Rusage
249 }
```

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

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

```go
500  }
501  func showFileInfo(path string, opts []string) {
502      if isin("-l",opts) || isin("-ls",opts) {
503          fi, err := os.Stat(path)
504          if err != nil {
505              fmt.Printf("---------- ((%v))",err)
506          }else{
507              mod := fi.ModTime()
508              date := mod.Format(time.Stamp)
509              fmt.Printf("%v %8v %s ",fi.Mode(),fi.Size(),date)
510          }
511      }
512      fmt.Printf("%s",path)
513      if isin("-sp",opts) {
514          fmt.Printf(" ")
515      }else
516      if ! isin("-n",opts) {
517          fmt.Printf("\n")
518      }
519  }
520  func userHomeDir()(string,bool){
521      /*
522      homedir,_ = os.UserHomeDir() // not implemented in older Golang
523      */
524      homedir,found := os.LookupEnv("HOME")
525      //fmt.Printf("--I-- HOME=%v(%v)\n",homedir,found)
526      if !found {
527          return "/tmp",found
528      }
529      return homedir,found
530  }
531
532  func toFullpath(path string) (fullpath string) {
533      if path[0] == '/' {
534          return path
535      }
536      pathv := strings.Split(path,DIRSEP)
537      switch {
538      case pathv[0] == ".":
539          pathv[0], _ = os.Getwd()
540      case pathv[0] == "..": // all ones should be interpreted
541          cwd, _ := os.Getwd()
542          ppathv := strings.Split(cwd,DIRSEP)
543          pathv[0] = strings.Join(ppathv,DIRSEP)
544      case pathv[0] == "~":
545          pathv[0],_ = userHomeDir()
546      default:
547          cwd, _ := os.Getwd()
548          pathv[0] = cwd + DIRSEP + pathv[0]
549      }
550      return strings.Join(pathv,DIRSEP)
551  }
552
553  func IsRegFile(path string)(bool){
554      fi, err := os.Stat(path)
555      if err == nil {
556          fm := fi.Mode()
557          return fm.IsRegular();
558      }
559      return false
560  }
561
562  // <a name="encode">Encode / Decode</a>
563  // <a href="https://golang.org/pkg/encoding/base64/#example_NewEncoder">Encoder</a>
564  func (gshCtx *GshContext)Enc(argv[]string){
565      file := os.Stdin
566      buff := make([]byte,LINESIZE)
567      li := 0
568      encoder := base64.NewEncoder(base64.StdEncoding,os.Stdout)
569      for li = 0; ; li++ {
570          count, err := file.Read(buff)
571          if count <= 0 {
572              break
573          }
574          if err != nil {
575              break
576          }
577          encoder.Write(buff[0:count])
578      }
579      encoder.Close()
580  }
581  func (gshCtx *GshContext)Dec(argv[]string){
582      decoder := base64.NewDecoder(base64.StdEncoding,os.Stdin)
583      li := 0
584      buff := make([]byte,LINESIZE)
585      for li = 0; ; li++ {
586          count, err := decoder.Read(buff)
587          if count <= 0 {
588              break
589          }
590          if err != nil {
591              break
592          }
593          os.Stdout.Write(buff[0:count])
594      }
595  }
596  // lnsp [N] [-crlf][-C \\]
597  func (gshCtx *GshContext)SplitLine(argv[]string){
598      reader := bufio.NewReaderSize(os.Stdin,64*1024)
599      ni := 0
600      toi := 0
601      for ni = 0; ; ni++ {
602          line, err := reader.ReadString('\n')
603          if len(line) <= 0 {
604              if err != nil {
605              fmt.Fprintf(os.Stderr,"--I-- lnsp %d to %d (%v)\n",ni,toi,err)
606              break
607              }
608          }
609          off := 0
610          ilen := len(line)
611          remlen := len(line)
612          for oi := 0; 0 < remlen; oi++ {
613              olen := remlen
614              addnl := false
615              if 72 < olen {
616                  olen = 72
617                  addnl = true
618              }
619              fmt.Fprintf(os.Stderr,"--D-- write %d [%d.%d] %d %d/%d/%d\n",
620                  toi,ni,oi,off,olen,remlen,ilen)
621              toi += 1
622              os.Stdout.Write([]byte(line[0:olen]))
623              if addnl {
624                  //os.Stdout.Write([]byte("\r\n"))
```

```
625                  os.Stdout.Write([]byte("\\"))
626                  os.Stdout.Write([]byte("\n"))
627              }
628              line = line[olen:]
629              off += olen
630              remlen -= olen
631          }
632      }
633      fmt.Fprintf(os.Stderr,"--I-- lnsp %d to %d\n",ni,toi)
634 }
635
636 // CRC32 <a href="http://golang.jp/pkg/hash-crc32">crc32</a>
637 // 1 0000 0100 1100 0001 0001 1101 1011 0111
638 var CRC32UNIX uint32 = uint32(0x04C11DB7) // Unix cksum
639 var CRC32IEEE uint32 = uint32(0xEDB88320)
640 func byteCRC32add(crc uint32,str[]byte,len uint64)(uint32){
641      var oi uint64
642      for oi = 0; oi < len; oi++ {
643          var oct = str[oi]
644          for bi := 0; bi < 8; bi++ {
645              //fprintf(stderr,"--CRC32 %d %X (%d.%d)\n",crc,oct,oi,bi)
646              ovf1 := (crc & 0x80000000) != 0
647              ovf2 := (oct & 0x80) != 0
648              ovf := (ovf1 && !ovf2) || (!ovf1 && ovf2)
649              oct <<= 1
650              crc <<= 1
651              if ovf { crc ^= CRC32UNIX }
652          }
653      }
654      //fprintf(stderr,"--CRC32 return %d %d\n",crc,len)
655      return crc;
656 }
657 func byteCRC32end(crc uint32, len uint64)(uint32){
658      var slen = make([]byte,4)
659      var li = 0
660          for li = 0; li < 4; {
661                  slen[li] = byte(len)
662          li += 1
663                  len >>= 8
664                  if( len == 0 ){
665                          break
666          }
667          }
668      crc = byteCRC32add(crc,slen,uint64(li))
669          crc ^= 0xFFFFFFFF
670          return crc
671 }
672 func strCRC32(str string,len uint64)(crc uint32){
673      crc = byteCRC32add(0,[]byte(str),len)
674      crc = byteCRC32end(crc,len)
675      //fprintf(stderr,"--CRC32 %d %d\n",crc,len)
676      return crc
677 }
678 func CRC32Finish(crc uint32, table *crc32.Table, len uint64)(uint32){
679      var slen = make([]byte,4)
680      var li = 0
681          for li = 0; li < 4; {
682                  slen[li] = byte(len & 0xFF)
683          li += 1
684                  len >>= 8
685                  if( len == 0 ){
686                          break
687          }
688          }
689      crc = crc32.Update(crc,table,slen)
690          crc ^= 0xFFFFFFFF
691          return crc
692 }
693
694 func (gsh*GshContext)xCksum(path string,argv[]string, sum*CheckSum)(int64){
695      if isin("-type/f",argv) && !IsRegFile(path){
696          return 0
697      }
698      if isin("-type/d",argv) && IsRegFile(path){
699          return 0
700      }
701      file, err := os.OpenFile(path,os.O_RDONLY,0)
702      if err != nil {
703          fmt.Printf("--E-- cksum %v (%v)\n",path,err)
704          return -1
705      }
706      defer file.Close()
707      if gsh.CmdTrace { fmt.Printf("--I-- cksum %v %v\n",path,argv) }
708
709      bi := 0
710      var buff = make([]byte,32*1024)
711      var total int64 = 0
712      var initTime = time.Time{}
713      if sum.Start == initTime {
714          sum.Start = time.Now()
715      }
716      for bi = 0; ; bi++ {
717          count,err := file.Read(buff)
718          if count <= 0 || err != nil {
719              break
720          }
721          if (sum.SumType & SUM_SUM64) != 0 {
722              s := sum.Sum64
723              for _,c := range buff[0:count] {
724                  s += uint64(c)
725              }
726              sum.Sum64 = s
727          }
728          if (sum.SumType & SUM_UNIXFILE) != 0 {
729              sum.Crc32Val = byteCRC32add(sum.Crc32Val,buff,uint64(count))
730          }
731          if (sum.SumType & SUM_CRCIEEE) != 0 {
732              sum.Crc32Val = crc32.Update(sum.Crc32Val,&sum.Crc32Table,buff[0:count])
733          }
734          // <a href="https://en.wikipedia.org/wiki/BSD_checksum">BSD checksum</a>
735          if (sum.SumType & SUM_SUM16_BSD) != 0 {
736              s := sum.Sum16
737              for _,c := range buff[0:count] {
738                  s = (s >> 1) + ((s & 1) << 15)
739                  s += int(c)
740                  s &= 0xFFFF
741                  //fmt.Printf("BSDsum: %d[%d] %d\n",sum.Size+int64(i),i,s)
742              }
743              sum.Sum16 = s
744          }
745          if (sum.SumType & SUM_SUM16_SYSV) != 0 {
746              for bj := 0; bj < count; bj++ {
747                  sum.Sum16 += int(buff[bj])
748              }
749          }
```

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

```
875        fsum.Size += fstat.Size
876        fsum.Blocks += int64(fstat.Blocks)
877        //if verb { fmt.Printf("(%8dBlk) %s",fstat.Blocks/2,path) }
878        if isin("-ls",argv){
879            //if verb { fmt.Printf("%4d %8d ",fstat.Blksize,fstat.Blocks) }
880 //          fmt.Printf("%d\t",fstat.Blocks/2)
881        }
882        //if finfo.IsDir()
883        if (fstat.Mode & S_IFMT) == S_IFDIR {
884            fsum.Dirs += 1
885        }
886        //if (finfo.Mode() & os.ModeSymlink) != 0
887        if (fstat.Mode & S_IFMT) == S_IFLNK {
888            //if verb { fmt.Printf("symlink(%v,%s)\n",fstat.Mode,finfo.Name()) }
889            //{ fmt.Printf("symlink(%o,%s)\n",fstat.Mode,finfo.Name()) }
890            fsum.SymLink += 1
891        }
892        return fsum
893 }
894 func (gsh*GshContext)xxFindEntv(depth int,total *fileSum,dir string, dstat syscall.Stat_t, ei int, entv []string,npatv[]string,argv[]string)(*fileSum){
895        nols := isin("-grep",argv)
896        // sort entv
897        /*
898        if isin("-t",argv){
899            sort.Slice(filev, func(i,j int) bool {
900                return 0 < filev[i].ModTime().Sub(filev[j].ModTime())
901            })
902        }
903        */
904            /*
905            if isin("-u",argv){
906                sort.Slice(filev, func(i,j int) bool {
907                    return 0 < filev[i].AccTime().Sub(filev[j].AccTime())
908                })
909            }
910            if isin("-U",argv){
911                sort.Slice(filev, func(i,j int) bool {
912                    return 0 < filev[i].CreatTime().Sub(filev[j].CreatTime())
913                })
914            }
915            */
916        /*
917        if isin("-S",argv){
918            sort.Slice(filev, func(i,j int) bool {
919                return filev[j].Size() < filev[i].Size()
920            })
921        }
922        */
923        for _,filename := range entv {
924            for _,npat := range npatv {
925                match := true
926                if npat == "*" {
927                    match = true
928                }else{
929                    match, _ = filepath.Match(npat,filename)
930                }
931                path := dir + DIRSEP + filename
932                if !match {
933                    continue
934                }
935                var fstat syscall.Stat_t
936                staterr := syscall.Lstat(path,&fstat)
937                if staterr != nil {
938                    if !isin("-w",argv){fmt.Printf("ufind: %v\n",staterr) }
939                    continue;
940                }
941                if isin("-du",argv) && (fstat.Mode & S_IFMT) == S_IFDIR {
942                    // should not show size of directory in "-du" mode ...
943                }else
944                if !nols && !isin("-s",argv) && (!isin("-du",argv) || isin("-a",argv)) {
945                    if isin("-du",argv) {
946                        fmt.Printf("%d\t",fstat.Blocks/2)
947                    }
948                    showFileInfo(path,argv)
949                }
950                if true { // && isin("-du",argv)
951                    total = cumFinfo(total,path,staterr,fstat,argv,false)
952                }
953                /*
954                if isin("-wc",argv) {
955                }
956                */
957                if gsh.lastCheckSum.SumType != 0 {
958                    gsh.xCksum(path,argv,&gsh.lastCheckSum);
959                }
960                x := isinX("-grep",argv); // -grep will be convenient like -ls
961                if 0 <= x && x+1 <= len(argv) { // -grep will be convenient like -ls
962                    if IsRegFile(path){
963                        found := gsh.xGrep(path,argv[x+1:])
964                        if 0 < found {
965                            foundv := gsh.CmdCurrent.FoundFile
966                            if len(foundv) < 10 {
967                                gsh.CmdCurrent.FoundFile =
968                                append(gsh.CmdCurrent.FoundFile,path)
969                            }
970                        }
971                    }
972                }
973                if !isin("-r0",argv) { // -d 0 in du, -depth n in find
974                    //total.Depth += 1
975                    if (fstat.Mode & S_IFMT) == S_IFLNK {
976                        continue
977                    }
978                    if dstat.Rdev != fstat.Rdev {
979                        fmt.Printf("--I-- don't follow differnet device %v(%v) %v(%v)\n",
980                            dir,dstat.Rdev,path,fstat.Rdev)
981                    }
982                    if (fstat.Mode & S_IFMT) == S_IFDIR {
983                        total = gsh.xxFind(depth+1,total,path,npatv,argv)
984                    }
985                }
986            }
987        }
988        return total
989 }
990 func (gsh*GshContext)xxFind(depth int,total *fileSum,dir string,npatv[]string,argv[]string)(*fileSum){
991        nols := isin("-grep",argv)
992        dirfile,oerr := os.OpenFile(dir,os.O_RDONLY,0)
993        if oerr == nil {
994            //fmt.Printf("--I-- %v(%v)[%d]\n",dir,dirfile,dirfile.Fd())
995            defer dirfile.Close()
996        }else{
997        }
998
999        prev := *total
```

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

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

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

```
1375                    }
1376                    arg1[ax] = ch
1377                    ax += 1
1378                    continue
1379                }
1380                if str[si] == ' ' {
1381                    argv = append(argv,string(arg1[0:ax]))
1382                    if debug {
1383                        fmt.Printf("--Da- [%v][%v-%v] %s ... %s\n",
1384                            -1+len(argv),sb,si,str[sb:si],string(str[si:]))
1385                    }
1386                    sb = si+1
1387                    ax = 0
1388                    continue
1389                }
1390                arg1[ax] = ch
1391                ax += 1
1392            }
1393            if sb < si {
1394                argv = append(argv,string(arg1[0:ax]))
1395                if debug {
1396                    fmt.Printf("--Da- [%v][%v-%v] %s ... %s\n",
1397                        -1+len(argv),sb,si,string(arg1[0:ax]),string(str[si:]))
1398                }
1399            }
1400            if debug {
1401                fmt.Printf("--Da- %d [%s] => [%d]%v\n",strip,str,len(argv),argv)
1402            }
1403            return argv
1404 }
1405
1406 // should get stderr (into tmpfile ?) and return
1407 func (gsh*GshContext)Popen(name,mode string)(pin*os.File,pout*os.File,err bool){
1408        var pv = []int{-1,-1}
1409        syscall.Pipe(pv)
1410
1411        xarg := gshScanArg(name,1)
1412        name = strings.Join(xarg," ")
1413
1414        pin = os.NewFile(uintptr(pv[0]),"StdoutOf-{"+name+"}")
1415        pout = os.NewFile(uintptr(pv[1]),"StdinOf-{"+name+"}")
1416        fdix := 0
1417        dir := "?"
1418        if mode == "r" {
1419            dir = "<"
1420            fdix = 1 // read from the stdout of the process
1421        }else{
1422            dir = ">"
1423            fdix = 0 // write to the stdin of the process
1424        }
1425        gshPA := gsh.gshPA
1426        savfd := gshPA.Files[fdix]
1427
1428        var fd uintptr = 0
1429        if mode == "r" {
1430            fd = pout.Fd()
1431            gshPA.Files[fdix] = pout.Fd()
1432        }else{
1433            fd = pin.Fd()
1434            gshPA.Files[fdix] = pin.Fd()
1435        }
1436            // should do this by Goroutine?
1437            if false {
1438                fmt.Printf("--Ip- Opened fd[%v] %s %v\n",fd,dir,name)
1439                fmt.Printf("--RED1 [%d,%d,%d]->[%d,%d,%d]\n",
1440                    os.Stdin.Fd(),os.Stdout.Fd(),os.Stderr.Fd(),
1441                    pin.Fd(),pout.Fd(),pout.Fd())
1442            }
1443                savi := os.Stdin
1444                savo := os.Stdout
1445                save := os.Stderr
1446                os.Stdin  = pin
1447                os.Stdout = pout
1448                os.Stderr = pout
1449            gsh.BackGround = true
1450            gsh.gshelllh(name)
1451            gsh.BackGround = false
1452                os.Stdin  = savi
1453                os.Stdout = savo
1454                os.Stderr = save
1455
1456        gshPA.Files[fdix] = savfd
1457        return pin,pout,false
1458 }
1459
1460 // <a name="ex-commands">External commands</a>
1461 func (gsh*GshContext)excommand(exec bool, argv []string) (notf bool,exit bool) {
1462        if gsh.CmdTrace { fmt.Printf("--I-- excommand[%v](%v)\n",exec,argv) }
1463
1464        gshPA := gsh.gshPA
1465        fullpathv, itis := which("PATH",[]string{"which",argv[0],"-s"})
1466        if itis == false {
1467            return true,false
1468        }
1469        fullpath := fullpathv[0]
1470        argv = unescapeWhiteSPV(argv)
1471        if 0 < strings.Index(fullpath,".go") {
1472            nargv := argv // []string{}
1473            gofullpathv, itis := which("PATH",[]string{"which","go","-s"})
1474            if itis == false {
1475                fmt.Printf("--F-- Go not found\n")
1476                return false,true
1477            }
1478            gofullpath := gofullpathv[0]
1479            nargv = []string{ gofullpath, "run", fullpath }
1480            fmt.Printf("--I-- %s {%s %s %s}\n",gofullpath,
1481                nargv[0],nargv[1],nargv[2])
1482            if exec {
1483                syscall.Exec(gofullpath,nargv,os.Environ())
1484            }else{
1485                pid, _ := syscall.ForkExec(gofullpath,nargv,&gshPA)
1486                if gsh.BackGround {
1487                    fmt.Fprintf(stderr,"--Ip- in Background pid[%d]%d(%v)\n",pid,len(argv),nargv)
1488                    gsh.BackGroundJobs = append(gsh.BackGroundJobs,pid)
1489                }else{
1490                    rusage := syscall.Rusage {}
1491                    syscall.Wait4(pid,nil,0,&rusage)
1492                    gsh.LastRusage = rusage
1493                    gsh.CmdCurrent.Rusagev[1] = rusage
1494                }
1495            }
1496        }else{
1497            if exec {
1498                syscall.Exec(fullpath,argv,os.Environ())
1499            }else{
```

```
1500                         pid, _ := syscall.ForkExec(fullpath,argv,&gshPA)
1501                         //fmt.Printf("[%d]\n",pid); // '&' to be background
1502                         if gsh.BackGround {
1503                                 fmt.Fprintf(stderr,"--Ip- in Background pid[%d]%d(%v)\n",pid,len(argv),argv)
1504                                 gsh.BackGroundJobs = append(gsh.BackGroundJobs,pid)
1505                         }else{
1506                                 rusage := syscall.Rusage {}
1507                                 syscall.Wait4(pid,nil,0,&rusage);
1508                                 gsh.LastRusage = rusage
1509                                 gsh.CmdCurrent.Rusagev[1] = rusage
1510                         }
1511                 }
1512         }
1513         return false,false
1514 }
1515
1516 // <a name="builtin">Builtin Commands</a>
1517 func (gshCtx *GshContext) sleep(argv []string) {
1518         if len(argv) < 2 {
1519                 fmt.Printf("Sleep 100ms, 100us, 100ns, ...\n")
1520                 return
1521         }
1522         duration := argv[1];
1523         d, err := time.ParseDuration(duration)
1524         if err != nil {
1525                 d, err = time.ParseDuration(duration+"s")
1526                 if err != nil {
1527                         fmt.Printf("duration ? %s (%s)\n",duration,err)
1528                         return
1529                 }
1530         }
1531         //fmt.Printf("Sleep %v\n",duration)
1532         time.Sleep(d)
1533         if 0 < len(argv[2:]) {
1534                 gshCtx.gshellv(argv[2:])
1535         }
1536 }
1537 func (gshCtx *GshContext)repeat(argv []string) {
1538         if len(argv) < 2 {
1539                 return
1540         }
1541         start0 := time.Now()
1542         for ri,_ := strconv.Atoi(argv[1]); 0 < ri; ri-- {
1543                 if 0 < len(argv[2:]) {
1544                         //start := time.Now()
1545                         gshCtx.gshellv(argv[2:])
1546                         end := time.Now()
1547                         elps := end.Sub(start0);
1548                         if( 1000000000 < elps ){
1549                                 fmt.Printf("(repeat#%d %v)\n",ri,elps);
1550                         }
1551                 }
1552         }
1553 }
1554
1555 func (gshCtx *GshContext)gen(argv []string) {
1556         gshPA := gshCtx.gshPA
1557         if len(argv) < 2 {
1558                 fmt.Printf("Usage: %s N\n",argv[0])
1559                 return
1560         }
1561         // should br repeated by "repeat" command
1562         count, _ := strconv.Atoi(argv[1])
1563         fd := gshPA.Files[1] // Stdout
1564         file := os.NewFile(fd,"internalStdOut")
1565         fmt.Printf("--I-- Gen. Count=%d to [%d]\n",count,file.Fd())
1566         //buf := []byte{}
1567         outdata := "0123 5678 0123 5678 0123 5678 0123 5678\r"
1568         for gi := 0; gi < count; gi++ {
1569                 file.WriteString(outdata)
1570         }
1571         //file.WriteString("\n")
1572         fmt.Printf("\n(%d B)\n",count*len(outdata));
1573         //file.Close()
1574 }
1575
1576 // <a name="rexec">Remote Execution</a> // 2020-0820
1577 func Elapsed(from time.Time)(string){
1578         elps := time.Now().Sub(from)
1579         if 1000000000 < elps {
1580                 return fmt.Sprintf("[%5d.%02ds]",elps/1000000000,(elps%1000000000)/10000000)
1581         }else
1582         if 1000000 < elps {
1583                 return fmt.Sprintf("[%3d.%03dms]",elps/1000000,(elps%1000000)/1000)
1584         }else{
1585                 return fmt.Sprintf("[%3d.%03dus]",elps/1000,(elps%1000))
1586         }
1587 }
1588 func abbtime(nanos int64)(string){
1589         if 1000000000 < nanos {
1590                 return fmt.Sprintf("%d.%02ds",nanos/1000000000,(nanos%1000000000)/10000000)
1591         }else
1592         if 1000000 < nanos {
1593                 return fmt.Sprintf("%d.%03dms",nanos/1000000,(nanos%1000000)/1000)
1594         }else{
1595                 return fmt.Sprintf("%d.%03dus",nanos/1000,(nanos%1000))
1596         }
1597 }
1598 func abssize(size int64)(string){
1599         fsize := float64(size)
1600         if 1024*1024*1024 < size {
1601                 return fmt.Sprintf("%.2fGiB",fsize/(1024*1024*1024))
1602         }else
1603         if 1024*1024 < size {
1604                 return fmt.Sprintf("%.3fMiB",fsize/(1024*1024))
1605         }else{
1606                 return fmt.Sprintf("%.3fKiB",fsize/1024)
1607         }
1608 }
1609 func absize(size int64)(string){
1610         fsize := float64(size)
1611         if 1024*1024*1024 < size {
1612                 return fmt.Sprintf("%8.2fGiB",fsize/(1024*1024*1024))
1613         }else
1614         if 1024*1024 < size {
1615                 return fmt.Sprintf("%8.3fMiB",fsize/(1024*1024))
1616         }else{
1617                 return fmt.Sprintf("%8.3fKiB",fsize/1024)
1618         }
1619 }
1620 func abbspeed(totalB int64,ns int64)(string){
1621         MBs := (float64(totalB)/1000000) / (float64(ns)/1000000000)
1622         if 1000 <= MBs {
1623                 return fmt.Sprintf("%6.3fGB/s",MBs/1000)
1624         }
```

```
1625            if 1 <= MBs {
1626                return fmt.Sprintf("%6.3fMB/s",MBs)
1627            }else{
1628                return fmt.Sprintf("%6.3fKB/s",MBs*1000)
1629            }
1630    }
1631    func abspeed(totalB int64,ns time.Duration)(string){
1632            MBs := (float64(totalB)/1000000) / (float64(ns)/1000000000)
1633            if 1000 <= MBs {
1634                return fmt.Sprintf("%6.3fGBps",MBs/1000)
1635            }
1636            if 1 <= MBs {
1637                return fmt.Sprintf("%6.3fMBps",MBs)
1638            }else{
1639                return fmt.Sprintf("%6.3fKBps",MBs*1000)
1640            }
1641    }
1642    func fileRelay(what string,in*os.File,out*os.File,size int64,bsiz int)(wcount int64){
1643            Start := time.Now()
1644            buff := make([]byte,bsiz)
1645            var total int64 = 0
1646            var rem int64 = size
1647            nio := 0
1648            Prev := time.Now()
1649            var PrevSize int64 = 0
1650
1651            fmt.Printf(Elapsed(Start)+"--In- X: %s (%v/%v/%v) START\n",
1652                what,absize(total),size,nio)
1653
1654            for i:= 0; ; i++ {
1655                var len = bsiz
1656                if int(rem) < len {
1657                    len = int(rem)
1658                }
1659                Now := time.Now()
1660                Elps := Now.Sub(Prev);
1661                if 1000000000 < Now.Sub(Prev) {
1662                    fmt.Printf(Elapsed(Start)+"--In- X: %s (%v/%v/%v) %s\n",
1663                        what,absize(total),size,nio,
1664                        abspeed((total-PrevSize),Elps))
1665                    Prev = Now;
1666                    PrevSize = total
1667                }
1668                rlen := len
1669                if in != nil {
1670                    // should watch the disconnection of out
1671                    rcc,err := in.Read(buff[0:rlen])
1672                    if err != nil {
1673                        fmt.Printf(Elapsed(Start)+"--En- X: %s read(%v,%v)<%v\n",
1674                            what,rcc,err,in.Name())
1675                        break
1676                    }
1677                    rlen = rcc
1678                    if string(buff[0:10]) == "((SoftEOF " {
1679                        var ecc int64 = 0
1680                        fmt.Sscanf(string(buff),"((SoftEOF %v",&ecc)
1681                        fmt.Printf(Elapsed(Start)+"--En- X: %s Recv ((SoftEOF %v))/%v\n",
1682                            what,ecc,total)
1683                        if ecc == total {
1684                            break
1685                        }
1686                    }
1687                }
1688
1689                wlen := rlen
1690                if out != nil {
1691                    wcc,err := out.Write(buff[0:rlen])
1692                    if err != nil {
1693                        fmt.Printf(Elapsed(Start)+"-En-- X: %s write(%v,%v)>%v\n",
1694                            what,wcc,err,out.Name())
1695                        break
1696                    }
1697                    wlen = wcc
1698                }
1699                if wlen < rlen {
1700                    fmt.Printf(Elapsed(Start)+"--En- X: %s incomplete write (%v/%v)\n",
1701                        what,wlen,rlen)
1702                    break;
1703                }
1704
1705                nio += 1
1706                total += int64(rlen)
1707                rem -= int64(rlen)
1708                if rem <= 0 {
1709                    break
1710                }
1711            }
1712            Done := time.Now()
1713            Elps := float64(Done.Sub(Start))/1000000000 //Seconds
1714            TotalMB := float64(total)/1000000 //MB
1715            MBps := TotalMB / Elps
1716            fmt.Printf(Elapsed(Start)+"--In- X: %s (%v/%v/%v) %v %.3fMB/s\n",
1717                what,total,size,nio,absize(total),MBps)
1718            return total
1719    }
1720    func tcpPush(clnt *os.File){
1721            // shrink socket buffer and recover
1722            usleep(100);
1723    }
1724    func (gsh*GshContext)RexecServer(argv[]string){
1725            debug := true
1726            Start0 := time.Now()
1727            Start := Start0
1728    //      if local == ":" { local = "0.0.0.0:9999" }
1729            local := "0.0.0.0:9999"
1730
1731            if 0 < len(argv) {
1732                if argv[0] == "-s" {
1733                    debug = false
1734                    argv = argv[1:]
1735                }
1736            }
1737            if 0 < len(argv) {
1738                argv = argv[1:]
1739            }
1740            port, err := net.ResolveTCPAddr("tcp",local);
1741            if err != nil {
1742                fmt.Printf("--En- S: Address error: %s (%s)\n",local,err)
1743                return
1744            }
1745            fmt.Printf(Elapsed(Start)+"--In- S: Listening at %s...\n",local);
1746            sconn, err := net.ListenTCP("tcp", port)
1747            if err != nil {
1748                fmt.Printf(Elapsed(Start)+"--En- S: Listen error: %s (%s)\n",local,err)
1749                return
```

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

```
1875                        fileRelay("RecvPUT",clnt,out,dsize,bsize)
1876                        res = fmt.Sprintf("200 PUT done\r\n")
1877                    default:
1878                        res = fmt.Sprintf("400 What? %v",req)
1879                    }
1880                    swcc,serr := clnt.Write([]byte(res))
1881                    if serr != nil {
1882                        fmt.Printf(Elapsed(Start)+"--In- S: (wc=%v er=%v) %v",swcc,serr,res)
1883                    }else{
1884                        fmt.Printf(Elapsed(Start)+"--In- S: %v",res)
1885                    }
1886                    aconn.Close();
1887                    clnt.Close();
1888        }
1889        sconn.Close();
1890 }
1891 func (gsh*GshContext)RexecClient(argv[]string)(int,string){
1892        debug := true
1893        Start := time.Now()
1894        if len(argv) == 1 {
1895            return -1,"EmptyARG"
1896        }
1897        argv = argv[1:]
1898        if argv[0] == "-serv" {
1899            gsh.RexecServer(argv[1:])
1900            return 0,"Server"
1901        }
1902        remote := "0.0.0.0:9999"
1903        if argv[0][0] == '@' {
1904            remote = argv[0][1:]
1905            argv = argv[1:]
1906        }
1907        if argv[0] == "-s" {
1908            debug = false
1909            argv = argv[1:]
1910        }
1911        dport, err := net.ResolveTCPAddr("tcp",remote);
1912        if err != nil {
1913            fmt.Printf(Elapsed(Start)+"Address error: %s (%s)\n",remote,err)
1914            return -1,"AddressError"
1915        }
1916        fmt.Printf(Elapsed(Start)+"--In- C: Connecting to %s\n",remote)
1917        serv, err := net.DialTCP("tcp",nil,dport)
1918        if err != nil {
1919            fmt.Printf(Elapsed(Start)+"Connection error: %s (%s)\n",remote,err)
1920            return -1,"CannotConnect"
1921        }
1922        if debug {
1923            al := serv.LocalAddr()
1924            fmt.Printf(Elapsed(Start)+"--In- C: Connected to %v <- %v\n",remote,al)
1925        }
1926
1927        req := ""
1928        res := make([]byte,LINESIZE)
1929        count,err := serv.Read(res)
1930        if err != nil {
1931            fmt.Printf("--En- S: (%3d,%v) %v",count,err,string(res))
1932        }
1933        if debug { fmt.Printf(Elapsed(Start)+"--In- S: %v",string(res)) }
1934
1935        if argv[0] == "GET" {
1936            savPA := gsh.gshPA
1937            var bsize int = 64*1024
1938            req = fmt.Sprintf("%v\r\n",strings.Join(argv," "))
1939            fmt.Printf(Elapsed(Start)+"--In- C: %v",req)
1940            fmt.Fprintf(serv,req)
1941            count,err = serv.Read(res)
1942            if err != nil {
1943            }else{
1944                var dsize int64 = 0
1945                var out *os.File = nil
1946                var out_tobeclosed *os.File = nil
1947                var fname string = ""
1948                var rcode int = 0
1949                var pid int = -1
1950                fmt.Sscanf(string(res),"%d %d",&rcode,&dsize)
1951                fmt.Printf(Elapsed(Start)+"--In- S: %v",string(res[0:count]))
1952                if 3 <= len(argv) {
1953                    fname = argv[2]
1954                    if strBegins(fname,"{") {
1955                        xin,xout,err := gsh.Popen(fname,"w")
1956                        if err {
1957                        }else{
1958                            xin.Close()
1959                            defer xout.Close()
1960                            out = xout
1961                            out_tobeclosed = xout
1962                            pid = 0 // should be its pid
1963                        }
1964                    }else{
1965                        // should write to temporary file
1966                        // should suppress ^C on tty
1967                        xout,err := os.OpenFile(fname,os.O_CREATE|os.O_RDWR|os.O_TRUNC,0600)
1968                        if err != nil {
1969                            fmt.Print("--En- %v\n",err)
1970                        }
1971                        out = xout
1972                        //fmt.Printf("--In-- %d > %s\n",out.Fd(),fname)
1973                    }
1974                }
1975                in,_ := serv.File()
1976                fileRelay("RecvGET",in,out,dsize,bsize)
1977                if 0 <= pid {
1978                    gsh.gshPA = savPA // recovery of Fd(), and more?
1979                    fmt.Printf(Elapsed(Start)+"--In- L: close Pipe > %v\n",fname)
1980                    out_tobeclosed.Close()
1981                    //syscall.Wait4(pid,nil,0,nil) //@@
1982                }
1983            }
1984        }else
1985        if argv[0] == "PUT" {
1986            remote, _ := serv.File()
1987            var local *os.File = nil
1988            var dsize int64 = 32*1024*1024
1989            var bsize int = 64*1024
1990            var ofile string = "-"
1991            //fmt.Printf("--I-- Rex %v\n",argv)
1992            if 1 < len(argv) {
1993                fname := argv[1]
1994                if strBegins(fname,"-z") {
1995                    fmt.Sscanf(fname[2:],"%d",&dsize)
1996                }else
1997                if strBegins(fname,"{") {
1998                    xin,xout,err := gsh.Popen(fname,"r")
1999                    if err {
```

```
2000                        }else{
2001                            xout.Close()
2002                            defer xin.Close()
2003                            //in = xin
2004                            local = xin
2005                            fmt.Printf("--In- [%d] < Upload output of %v\n",
2006                                local.Fd(),fname)
2007                            ofile = "-from."+fname
2008                            dsize = MaxStreamSize
2009                        }
2010                    }else{
2011                        xlocal,err := os.Open(fname)
2012                        if err != nil {
2013                            fmt.Printf("--En- (%s)\n",err)
2014                            local = nil
2015                        }else{
2016                            local = xlocal
2017                            fi,_ := local.Stat()
2018                            dsize = fi.Size()
2019                            defer local.Close()
2020                            //fmt.Printf("--I-- Rex in(%v / %v)\n",ofile,dsize)
2021                        }
2022                        ofile = fname
2023                        fmt.Printf(Elapsed(Start)+"--In- L: open(%v,r)=%v %v (%v)\n",
2024                            fname,dsize,local,err)
2025                    }
2026                }
2027                if 2 < len(argv) && argv[2] != "" {
2028                    ofile = argv[2]
2029                    //fmt.Printf("(%d)%v B.ofile=%v\n",len(argv),argv,ofile)
2030                }
2031                //fmt.Printf(Elapsed(Start)+"--I-- Rex out(%v)\n",ofile)
2032                fmt.Printf(Elapsed(Start)+"--In- PUT %v (/%v)\n",dsize,bsize)
2033                req = fmt.Sprintf("PUT %v %v \r\n",dsize,ofile)
2034                if debug { fmt.Printf(Elapsed(Start)+"--In- C: %v",req) }
2035                fmt.Fprintf(serv,"%v",req)
2036                count,err = serv.Read(res)
2037                if debug { fmt.Printf(Elapsed(Start)+"--In- S: %v",string(res[0:count])) }
2038                fileRelay("SendPUT",local,remote,dsize,bsize)
2039            }else{
2040                req = fmt.Sprintf("%v\r\n",strings.Join(argv," "))
2041                if debug { fmt.Printf(Elapsed(Start)+"--In- C: %v",req) }
2042                fmt.Fprintf(serv,"%v",req)
2043                //fmt.Printf("--In- sending RexRequest(%v)\n",len(req))
2044            }
2045            //fmt.Printf(Elapsed(Start)+"--In- waiting RexResponse...\n")
2046            count,err = serv.Read(res)
2047            ress := ""
2048            if count == 0 {
2049                ress = "(nil)\r\n"
2050            }else{
2051                ress = string(res[:count])
2052            }
2053            if err != nil {
2054                fmt.Printf(Elapsed(Start)+"--En- S: (%d,%v) %v",count,err,ress)
2055            }else{
2056                fmt.Printf(Elapsed(Start)+"--In- S: %v",ress)
2057            }
2058            serv.Close()
2059            //conn.Close()
2060
2061            var stat string
2062            var rcode int
2063            fmt.Sscanf(ress,"%d %s",&rcode,&stat)
2064            //fmt.Printf("--D-- Client: %v (%v)",rcode,stat)
2065            return rcode,ress
2066 }
2067
2068 // <a name="remote-sh">Remote Shell</a>
2069 // gcp file [...] { [host]:[port:][dir] | dir } // -p | -no-p
2070 func (gsh*GshContext)FileCopy(argv[]string){
2071     var host = ""
2072     var port = ""
2073     var upload = false
2074     var download = false
2075     var xargv = []string{"rex-gcp"}
2076     var srcv = []string{}
2077     var dstv = []string{}
2078     argv = argv[1:]
2079
2080     for _,v := range argv {
2081         /*
2082         if v[0] == '-' { // might be a pseudo file (generated date)
2083             continue
2084         }
2085         */
2086         obj := strings.Split(v,":")
2087         //fmt.Printf("%d %v %v\n",len(obj),v,obj)
2088         if 1 < len(obj) {
2089             host = obj[0]
2090             file := ""
2091             if 0 < len(host) {
2092                 gsh.LastServer.host = host
2093             }else{
2094                 host = gsh.LastServer.host
2095                 port = gsh.LastServer.port
2096             }
2097             if 2 < len(obj) {
2098                 port = obj[1]
2099                 if 0 < len(port) {
2100                     gsh.LastServer.port = port
2101                 }else{
2102                     port = gsh.LastServer.port
2103                 }
2104                 file = obj[2]
2105             }else{
2106                 file = obj[1]
2107             }
2108             if len(srcv) == 0 {
2109                 download = true
2110                 srcv = append(srcv,file)
2111                 continue
2112             }
2113             upload = true
2114             dstv = append(dstv,file)
2115             continue
2116         }
2117         /*
2118         idx := strings.Index(v,":")
2119         if 0 <= idx {
2120             remote = v[0:idx]
2121             if len(srcv) == 0 {
2122                 download = true
2123                 srcv = append(srcv,v[idx+1:])
2124                 continue
2125```

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

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

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

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

```
2625        TimeValAdd(ru1[0].Utime,ru2[0].Utime)
2626        TimeValAdd(ru1[0].Stime,ru2[0].Stime)
2627        TimeValAdd(ru1[1].Utime,ru2[1].Utime)
2628        TimeValAdd(ru1[1].Stime,ru2[1].Stime)
2629        return ru1
2630 }
2631 */
2632
2633 // <a name="rusage">Resource Usage</a>
2634 func sRusagef(fmtspec string, argv []string, ru [2]syscall.Rusage)(string){
2635        // ru[0] self , ru[1] children
2636        ut := TimeValAdd(ru[0].Utime,ru[1].Utime)
2637        st := TimeValAdd(ru[0].Stime,ru[1].Stime)
2638        uu := (ut.Sec*1000000 + int64(ut.Usec)) * 1000
2639        su := (st.Sec*1000000 + int64(st.Usec)) * 1000
2640        tu := uu + su
2641        ret := fmt.Sprintf("%v/sum",abbtime(tu))
2642        ret += fmt.Sprintf(", %v/usr",abbtime(uu))
2643        ret += fmt.Sprintf(", %v/sys",abbtime(su))
2644        return ret
2645 }
2646 func Rusagef(fmtspec string, argv []string, ru [2]syscall.Rusage)(string){
2647        ut := TimeValAdd(ru[0].Utime,ru[1].Utime)
2648        st := TimeValAdd(ru[0].Stime,ru[1].Stime)
2649        fmt.Printf("%d.%06ds/u ",ut.Sec,ut.Usec) //ru[1].Utime.Sec,ru[1].Utime.Usec)
2650        fmt.Printf("%d.%06ds/s ",st.Sec,st.Usec) //ru[1].Stime.Sec,ru[1].Stime.Usec)
2651        return ""
2652 }
2653 func Getrusagev()([2]syscall.Rusage){
2654        var ruv = [2]syscall.Rusage{}
2655        syscall.Getrusage(syscall.RUSAGE_SELF,&ruv[0])
2656        syscall.Getrusage(syscall.RUSAGE_CHILDREN,&ruv[1])
2657        return ruv
2658 }
2659 func showRusage(what string,argv []string, ru *syscall.Rusage){
2660        fmt.Printf("%s: ",what);
2661        fmt.Printf("Usr=%d.%06ds",ru.Utime.Sec,ru.Utime.Usec)
2662        fmt.Printf(" Sys=%d.%06ds",ru.Stime.Sec,ru.Stime.Usec)
2663        fmt.Printf(" Rss=%vB",ru.Maxrss)
2664        if isin("-l",argv) {
2665            fmt.Printf(" MinFlt=%v",ru.Minflt)
2666            fmt.Printf(" MajFlt=%v",ru.Majflt)
2667            fmt.Printf(" IxRSS=%vB",ru.Ixrss)
2668            fmt.Printf(" IdRSS=%vB",ru.Idrss)
2669            fmt.Printf(" Nswap=%vB",ru.Nswap)
2670        fmt.Printf(" Read=%v",ru.Inblock)
2671        fmt.Printf(" Write=%v",ru.Oublock)
2672        }
2673        fmt.Printf(" Snd=%v",ru.Msgsnd)
2674        fmt.Printf(" Rcv=%v",ru.Msgrcv)
2675        //if isin("-l",argv) {
2676            fmt.Printf(" Sig=%v",ru.Nsignals)
2677        //}
2678        fmt.Printf("\n");
2679 }
2680 func (gshCtx *GshContext)xTime(argv[]string)(bool){
2681        if 2 <= len(argv){
2682            gshCtx.LastRusage = syscall.Rusage{}
2683            rusagev1 := Getrusagev()
2684            fin := gshCtx.gshellv(argv[1:])
2685            rusagev2 := Getrusagev()
2686            showRusage(argv[1],argv,&gshCtx.LastRusage)
2687            rusagev := RusageSubv(rusagev2,rusagev1)
2688            showRusage("self",argv,&rusagev[0])
2689            showRusage("chld",argv,&rusagev[1])
2690            return fin
2691        }else{
2692            rusage:= syscall.Rusage {}
2693            syscall.Getrusage(syscall.RUSAGE_SELF,&rusage)
2694            showRusage("self",argv, &rusage)
2695            syscall.Getrusage(syscall.RUSAGE_CHILDREN,&rusage)
2696            showRusage("chld",argv, &rusage)
2697            return false
2698        }
2699 }
2700 func (gshCtx *GshContext)xJobs(argv[]string){
2701        fmt.Printf("%d Jobs\n",len(gshCtx.BackGroundJobs))
2702        for ji, pid := range gshCtx.BackGroundJobs {
2703            //wstat := syscall.WaitStatus {0}
2704            rusage := syscall.Rusage {}
2705            //wpid, err := syscall.Wait4(pid,&wstat,syscall.WNOHANG,&rusage);
2706            wpid, err := syscall.Wait4(pid,nil,syscall.WNOHANG,&rusage);
2707            if err != nil {
2708                fmt.Printf("--E-- %%%d [%d] (%v)\n",ji,pid,err)
2709            }else{
2710                fmt.Printf("%%%d[%d](%d)\n",ji,pid,wpid)
2711                showRusage("chld",argv,&rusage)
2712            }
2713        }
2714 }
2715 func (gsh*GshContext)inBackground(argv[]string)(bool){
2716        if gsh.CmdTrace { fmt.Printf("--I-- inBackground(%v)\n",argv) }
2717        gsh.BackGround = true // set background option
2718        xfin := false
2719        xfin = gsh.gshellv(argv)
2720        gsh.BackGround = false
2721        return xfin
2722 }
2723 // -o file without command means just opening it and refer by #N
2724 // should be listed by "files" comnnand
2725 func (gshCtx*GshContext)xOpen(argv[]string){
2726        var pv = []int{-1,-1}
2727        err := syscall.Pipe(pv)
2728        fmt.Printf("--I-- pipe()=[#%d,#%d](%v)\n",pv[0],pv[1],err)
2729 }
2730 func (gshCtx*GshContext)fromPipe(argv[]string){
2731 }
2732 func (gshCtx*GshContext)xClose(argv[]string){
2733 }
2734
2735 // <a name="redirect">redirect</a>
2736 func (gshCtx*GshContext)redirect(argv[]string)(bool){
2737        if len(argv) < 2 {
2738            return false
2739        }
2740
2741        cmd := argv[0]
2742        fname := argv[1]
2743        var file *os.File = nil
2744
2745        fdix := 0
2746        mode := os.O_RDONLY
2747
2748        switch {
2749        case cmd == "-i" || cmd == "<":
```

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

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

```
3000            n,err := fmt.Sscanf(input,fmts,&v[0],&v[1],&v[2],&v[3])
3001            fmt.Printf("--D-- Scanf ->(%v) n=%d err=(%v)\n",v,n,err)
3002            gsh.iValues = v
3003        }
3004 }
3005 func (gsh*GshContext)Printv(argv[]string){
3006        if false { //@@U
3007            fmt.Printf("%v\n",strings.Join(argv[1:]," "))
3008            return
3009        }
3010        //fmt.Printf("--D-- Printv(%v)\n",argv)
3011        //fmt.Printf("%v\n",strings.Join(gsh.iValues,","))
3012        div := gsh.iDelimiter
3013        fmts := ""
3014        argv = argv[1:]
3015        if 0 < len(argv) {
3016            if strBegins(argv[0],"-F") {
3017                div = argv[0][2:]
3018                argv = argv[1:]
3019            }
3020        }
3021
3022        optv := []string{}
3023        for _,v := range argv {
3024            if strBegins(v,"-"){
3025                optv = append(optv,v)
3026                argv = argv[1:]
3027            }else{
3028                break;
3029            }
3030        }
3031        if 0 < len(argv) {
3032            fmts = strings.Join(argv," ")
3033        }
3034        gsh.printfv(fmts,div,argv,optv,gsh.iValues)
3035 }
3036 func (gsh*GshContext)Basename(argv[]string){
3037        for i,v := range gsh.iValues {
3038            gsh.iValues[i] = filepath.Base(v)
3039        }
3040 }
3041 func (gsh*GshContext)Sortv(argv[]string){
3042        sv := gsh.iValues
3043        sort.Slice(sv , func(i,j int) bool {
3044            return sv[i] < sv[j]
3045        })
3046 }
3047 func (gsh*GshContext)Shiftv(argv[]string){
3048        vi := len(gsh.iValues)
3049        if 0 < vi {
3050            if isin("-r",argv) {
3051                top := gsh.iValues[0]
3052                gsh.iValues = append(gsh.iValues[1:],top)
3053            }else{
3054                gsh.iValues = gsh.iValues[1:]
3055            }
3056        }
3057 }
3058
3059 func (gsh*GshContext)Enq(argv[]string){
3060 }
3061 func (gsh*GshContext)Deq(argv[]string){
3062 }
3063 func (gsh*GshContext)Push(argv[]string){
3064        gsh.iValStack = append(gsh.iValStack,argv[1:])
3065        fmt.Printf("depth=%d\n",len(gsh.iValStack))
3066 }
3067 func (gsh*GshContext)Dump(argv[]string){
3068        for i,v := range gsh.iValStack {
3069            fmt.Printf("%d %v\n",i,v)
3070        }
3071 }
3072 func (gsh*GshContext)Pop(argv[]string){
3073        depth := len(gsh.iValStack)
3074        if 0 < depth {
3075            v := gsh.iValStack[depth-1]
3076            if isin("-cat",argv){
3077                gsh.iValues = append(gsh.iValues,v...)
3078            }else{
3079                gsh.iValues = v
3080            }
3081            gsh.iValStack = gsh.iValStack[0:depth-1]
3082            fmt.Printf("depth=%d %s\n",len(gsh.iValStack),gsh.iValues)
3083        }else{
3084            fmt.Printf("depth=%d\n",depth)
3085        }
3086 }
3087
3088 // <a name="interpreter">Command Interpreter</a>
3089 func (gshCtx*GshContext)gshellv(argv []string) (fin bool) {
3090        fin = false
3091
3092        if gshCtx.CmdTrace { fmt.Fprintf(os.Stderr,"--I-- gshellv((%d))\n",len(argv)) }
3093        if len(argv) <= 0 {
3094            return false
3095        }
3096        xargv := []string{}
3097        for ai := 0; ai < len(argv); ai++ {
3098            xargv = append(xargv,strsubst(gshCtx,argv[ai],false))
3099        }
3100        argv = xargv
3101        if false {
3102            for ai := 0; ai < len(argv); ai++ {
3103                fmt.Printf("[%d] %s [%d]%T\n",
3104                    ai,argv[ai],len(argv[ai]),argv[ai])
3105            }
3106        }
3107        cmd := argv[0]
3108        if gshCtx.CmdTrace { fmt.Fprintf(os.Stderr,"--I-- gshellv(%d)%v\n",len(argv),argv) }
3109        switch { // https://tour.golang.org/flowcontrol/11
3110        case cmd == "":
3111            gshCtx.xPwd([]string{}); // emtpy command
3112        case cmd == "-x":
3113            gshCtx.CmdTrace = ! gshCtx.CmdTrace
3114        case cmd == "-xt":
3115            gshCtx.CmdTime = ! gshCtx.CmdTime
3116        case cmd == "-ot":
3117            gshCtx.sconnect(true, argv)
3118        case cmd == "-ou":
3119            gshCtx.sconnect(false, argv)
3120        case cmd == "-it":
3121            gshCtx.saccept(true , argv)
3122        case cmd == "-iu":
3123            gshCtx.saccept(false, argv)
3124        case cmd == "-i" || cmd == "<" || cmd == "-o" || cmd == ">" || cmd == "-a" || cmd == ">>" || cmd == "-s" || cmd == "><":
```

```
3125            gshCtx.redirect(argv)
3126        case cmd == "|":
3127            gshCtx.fromPipe(argv)
3128        case cmd == "args":
3129            gshCtx.Args(argv)
3130        case cmd == "bg" || cmd == "-bg":
3131            rfin := gshCtx.inBackground(argv[1:])
3132            return rfin
3133        case cmd == "-bn":
3134            gshCtx.Basename(argv)
3135        case cmd == "call":
3136            _,_ = gshCtx.excommand(false,argv[1:])
3137        case cmd == "cd" || cmd == "chdir":
3138            gshCtx.xChdir(argv);
3139        case cmd == "-cksum":
3140            gshCtx.xFind(argv)
3141        case cmd == "-sum":
3142            gshCtx.xFind(argv)
3143        case cmd == "-sumtest":
3144            str := ""
3145            if 1 < len(argv) { str = argv[1] }
3146            crc := strCRC32(str,uint64(len(str)))
3147            fprintf(stderr,"%v %v\n",crc,len(str))
3148        case cmd == "close":
3149            gshCtx.xClose(argv)
3150        case cmd == "gcp":
3151            gshCtx.FileCopy(argv)
3152        case cmd == "dec" || cmd == "decode":
3153            gshCtx.Dec(argv)
3154        case cmd == "#define":
3155        case cmd == "dic" || cmd == "d":
3156            xDic(argv)
3157        case cmd == "dump":
3158            gshCtx.Dump(argv)
3159        case cmd == "echo" || cmd == "e":
3160            echo(argv,true)
3161        case cmd == "enc" || cmd == "encode":
3162            gshCtx.Enc(argv)
3163        case cmd == "env":
3164            env(argv)
3165        case cmd == "eval":
3166            xEval(argv[1:],true)
3167        case cmd == "ev" || cmd == "events":
3168            dumpEvents(argv)
3169        case cmd == "exec":
3170            _,_ = gshCtx.excommand(true,argv[1:])
3171            // should not return here
3172        case cmd == "exit" || cmd == "quit":
3173            // write Result code EXIT to 3>
3174            return true
3175        case cmd == "fdls":
3176            // dump the attributes of fds (of other process)
3177        case cmd == "-find" || cmd == "fin" || cmd == "ufind" || cmd == "uf":
3178            gshCtx.xFind(argv[1:])
3179        case cmd == "fu":
3180            gshCtx.xFind(argv[1:])
3181        case cmd == "fork":
3182            // mainly for a server
3183        case cmd == "-gen":
3184            gshCtx.gen(argv)
3185        case cmd == "-go":
3186            gshCtx.xGo(argv)
3187        case cmd == "-grep":
3188            gshCtx.xFind(argv)
3189        case cmd == "gdeq":
3190            gshCtx.Deq(argv)
3191        case cmd == "genq":
3192            gshCtx.Enq(argv)
3193        case cmd == "gpop":
3194            gshCtx.Pop(argv)
3195        case cmd == "gpush":
3196            gshCtx.Push(argv)
3197        case cmd == "history" || cmd == "hi": // hi should be alias
3198            gshCtx.xHistory(argv)
3199        case cmd == "jobs":
3200            gshCtx.xJobs(argv)
3201        case cmd == "lnsp" || cmd == "nlsp":
3202            gshCtx.SplitLine(argv)
3203        case cmd == "-ls":
3204            gshCtx.xFind(argv)
3205        case cmd == "nop":
3206            // do nothing
3207        case cmd == "pipe":
3208            gshCtx.xOpen(argv)
3209        case cmd == "plug" || cmd == "plugin" || cmd == "pin":
3210            gshCtx.xPlugin(argv[1:])
3211        case cmd == "print" || cmd == "-pr":
3212            // output internal slice // also sprintf should be
3213            gshCtx.Printv(argv)
3214        case cmd == "ps":
3215            gshCtx.xPs(argv)
3216        case cmd == "pstitle":
3217            // to be gsh.title
3218        case cmd == "rexecd" || cmd == "rexd":
3219            gshCtx.RexecServer(argv)
3220        case cmd == "rexec" || cmd == "rex":
3221            gshCtx.RexecClient(argv)
3222        case cmd == "repeat" || cmd == "rep": // repeat cond command
3223            gshCtx.repeat(argv)
3224        case cmd == "replay":
3225            gshCtx.xReplay(argv)
3226        case cmd == "scan":
3227            // scan input (or so in fscanf) to internal slice (like Files or map)
3228            gshCtx.Scanv(argv)
3229        case cmd == "set":
3230            // set name ...
3231        case cmd == "serv":
3232            gshCtx.httpServer(argv)
3233        case cmd == "shift":
3234            gshCtx.Shiftv(argv)
3235        case cmd == "sleep":
3236            gshCtx.sleep(argv)
3237        case cmd == "-sort":
3238            gshCtx.Sortv(argv)
3239
3240        case cmd == "j" || cmd == "join":
3241            gshCtx.Rjoin(argv)
3242        case cmd == "a" || cmd == "alpa":
3243            gshCtx.Rexec(argv)
3244        case cmd == "jcd" || cmd == "jchdir":
3245            gshCtx.Rchdir(argv)
3246        case cmd == "jget":
3247            gshCtx.Rget(argv)
3248        case cmd == "jls":
3249            gshCtx.Rls(argv)
```

```
3250          case cmd == "jput":
3251              gshCtx.Rput(argv)
3252          case cmd == "jpwd":
3253              gshCtx.Rpwd(argv)
3254
3255          case cmd == "time":
3256              fin = gshCtx.xTime(argv)
3257          case cmd == "ungets":
3258              if 1 < len(argv) {
3259                  ungets(argv[1]+"\n")
3260              }else{
3261              }
3262          case cmd == "pwd":
3263              gshCtx.xPwd(argv);
3264          case cmd == "ver" || cmd == "-ver" || cmd == "version":
3265              gshCtx.showVersion(argv)
3266          case cmd == "where":
3267              // data file or so?
3268          case cmd == "which":
3269              which("PATH",argv);
3270          default:
3271              if gshCtx.whichPlugin(cmd,[]string{"-s"}) != nil {
3272                  gshCtx.xPlugin(argv)
3273              }else{
3274                  notfound,_ := gshCtx.excommand(false,argv)
3275                  if notfound {
3276                      fmt.Printf("--E-- command not found (%v)\n",cmd)
3277                  }
3278              }
3279          }
3280          return fin
3281  }
3282
3283  func (gsh*GshContext)gshelll(gline string) (rfin bool) {
3284      argv := strings.Split(string(gline)," ")
3285      fin := gsh.gshellv(argv)
3286      return fin
3287  }
3288  func (gsh*GshContext)tgshelll(gline string)(xfin bool){
3289      start := time.Now()
3290      fin := gsh.gshelll(gline)
3291      end := time.Now()
3292      elps := end.Sub(start);
3293      if gsh.CmdTime {
3294          fmt.Printf("--T-- " + time.Now().Format(time.Stamp) + "(%d.%09ds)\n",
3295              elps/1000000000,elps%1000000000)
3296      }
3297      return fin
3298  }
3299  func Ttyid() (int) {
3300      fi, err := os.Stdin.Stat()
3301      if err != nil {
3302          return 0;
3303      }
3304      //fmt.Printf("Stdin: %v Dev=%d\n",
3305      //  fi.Mode(),fi.Mode()&os.ModeDevice)
3306      if (fi.Mode() & os.ModeDevice) != 0 {
3307          stat := syscall.Stat_t{};
3308          err := syscall.Fstat(0,&stat)
3309          if err != nil {
3310              //fmt.Printf("--I-- Stdin: (%v)\n",err)
3311          }else{
3312              //fmt.Printf("--I-- Stdin: rdev=%d %d\n",
3313              //  stat.Rdev&0xFF,stat.Rdev);
3314              //fmt.Printf("--I-- Stdin: tty%d\n",stat.Rdev&0xFF);
3315              return int(stat.Rdev & 0xFF)
3316          }
3317      }
3318      return 0
3319  }
3320  func (gshCtx *GshContext) ttyfile() string {
3321      //fmt.Printf("--I-- GSH_HOME=%s\n",gshCtx.GshHomeDir)
3322      ttyfile := gshCtx.GshHomeDir + "/" + "gsh-tty" +
3323          fmt.Sprintf("%02d",gshCtx.TerminalId)
3324          //strconv.Itoa(gshCtx.TerminalId)
3325      //fmt.Printf("--I-- ttyfile=%s\n",ttyfile)
3326      return ttyfile
3327  }
3328  func (gshCtx *GshContext) ttyline()(*os.File){
3329      file, err := os.OpenFile(gshCtx.ttyfile(),os.O_RDWR|os.O_CREATE|os.O_TRUNC,0600)
3330      if err != nil {
3331          fmt.Printf("--F-- cannot open %s (%s)\n",gshCtx.ttyfile(),err)
3332          return file;
3333      }
3334      return file
3335  }
3336  func (gshCtx *GshContext)getline(hix int, skipping bool, prevline string) (string) {
3337      if( skipping ){
3338          reader := bufio.NewReaderSize(os.Stdin,LINESIZE)
3339          line, _, _ := reader.ReadLine()
3340          return string(line)
3341      }else
3342      if true {
3343          return xgetline(hix,prevline,gshCtx)
3344      }
3345      /*
3346      else
3347      if( with_exgetline && gshCtx.GetLine != "" ){
3348          //var xhix int64 = int64(hix); // cast
3349          newenv := os.Environ()
3350          newenv = append(newenv, "GSH_LINENO="+strconv.FormatInt(int64(hix),10) )
3351
3352          tty := gshCtx.ttyline()
3353          tty.WriteString(prevline)
3354          Pa := os.ProcAttr {
3355              "", // start dir
3356              newenv, //os.Environ(),
3357              []*os.File{os.Stdin,os.Stdout,os.Stderr,tty},
3358              nil,
3359          }
3360  //fmt.Printf("--I-- getline=%s // %s\n",gsh_getlinev[0],gshCtx.GetLine)
3361  proc, err := os.StartProcess(gsh_getlinev[0],[]string{"getline","getline"},&Pa)
3362          if err != nil {
3363              fmt.Printf("--F-- getline process error (%v)\n",err)
3364              // for ; ; { }
3365              return "exit (getline program failed)"
3366          }
3367          //stat, err := proc.Wait()
3368          proc.Wait()
3369          buff := make([]byte,LINESIZE)
3370          count, err := tty.Read(buff)
3371          //_, err = tty.Read(buff)
3372          //fmt.Printf("--D-- getline (%d)\n",count)
3373          if err != nil {
3374              if ! (count == 0) { // && err.String() == "EOF" ) {
```

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

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

```
3625         DIC_COM_DUMP = "s"
3626         DIC_COM_LIST = "ls"
3627         DIC_COM_ENA  = "en"
3628         DIC_COM_DIS  = "di"
3629 )
3630 func helpDic(argv []string){
3631     out := stderr
3632     cmd := ""
3633     if 0 < len(argv) { cmd = argv[0] }
3634     fprintf(out,"--- %v Usage\n",cmd)
3635     fprintf(out,"... Commands\n")
3636     fprintf(out,"...   %v %-3v [dicName] [dicURL ] -- Import dictionary\n",cmd,DIC_COM_LOAD)
3637     fprintf(out,"...   %v %-3v [pattern] -- Search in dictionary\n",cmd,DIC_COM_DUMP)
3638     fprintf(out,"...   %v %-3v [dicName] -- List dictionaries\n",cmd,DIC_COM_LIST)
3639     fprintf(out,"...   %v %-3v [dicName] -- Disable dictionaries\n",cmd,DIC_COM_DIS)
3640     fprintf(out,"...   %v %-3v [dicName] -- Enable dictionaries\n",cmd,DIC_COM_ENA)
3641     fprintf(out,"... Keys ... %v\n","ESC can be used for '\\'")
3642     fprintf(out,"...   \\c -- Reverse the case of the last character\n",)
3643     fprintf(out,"...   \\i -- Replace input with translated text\n",)
3644     fprintf(out,"...   \\j -- On/Off translation mode\n",)
3645     fprintf(out,"...   \\l -- Force Lower Case\n",)
3646     fprintf(out,"...   \\u -- Force Upper Case (software CapsLock)\n",)
3647     fprintf(out,"...   \\v -- Show translation actions\n",)
3648     fprintf(out,"...   \\x -- Replace the last input character with it Hexa-Decimal\n",)
3649 }
3650 func xDic(argv[]string){
3651     if len(argv) <= 1 {
3652         helpDic(argv)
3653         return
3654     }
3655     argv = argv[1:]
3656     var debug = false
3657     var info = false
3658     var silent = false
3659     var dump = false
3660     var builtin = false
3661     cmd := argv[0]
3662     argv = argv[1:]
3663     opt := ""
3664     arg := ""
3665
3666     if 0 < len(argv) {
3667         arg1 := argv[0]
3668         if arg1[0] == '-' {
3669             switch arg1 {
3670                 default: fmt.Printf("--Ed-- Unknown option(%v)\n",arg1)
3671                     return
3672                 case "-b": builtin = true
3673                 case "-d": debug = true
3674                 case "-s": silent = true
3675                 case "-v": info = true
3676             }
3677             opt = arg1
3678             argv = argv[1:]
3679         }
3680     }
3681
3682     dicName := ""
3683     dicURL := ""
3684     if 0 < len(argv) {
3685         arg = argv[0]
3686         dicName = arg
3687         argv = argv[1:]
3688     }
3689     if 0 < len(argv) {
3690         dicURL = argv[0]
3691         argv = argv[1:]
3692     }
3693     if false {
3694         fprintf(stderr,"--Dd-- com(%v) opt(%v) arg(%v)\n",cmd,opt,arg)
3695     }
3696     if cmd == DIC_COM_LOAD {
3697         //dicType := ""
3698         dicBody := ""
3699         if !builtin && dicName != "" && dicURL == "" {
3700             f,err := os.Open(dicName)
3701             if err == nil {
3702                 dicURL = dicName
3703             }else{
3704                 f,err = os.Open(dicName+".html")
3705                 if err == nil {
3706                     dicURL = dicName+".html"
3707                 }else{
3708                     f,err = os.Open("gshdic-"+dicName+".html")
3709                     if err == nil {
3710                         dicURL = "gshdic-"+dicName+".html"
3711                     }
3712                 }
3713             }
3714             if err == nil {
3715                 var buf = make([]byte,128*1024)
3716                 count,err := f.Read(buf)
3717                 f.Close()
3718                 if info {
3719                     fprintf(stderr,"--Id-- ReadDic(%v,%v)\n",count,err)
3720                 }
3721                 dicBody = string(buf[0:count])
3722             }
3723         }
3724         if dicBody == "" {
3725             switch arg {
3726                 default:
3727                     dicName = "WorldDic"
3728                     dicURL = WorldDic
3729                     if info {
3730                         fprintf(stderr,"--Id-- default dictionary \"%v\"\n",
3731                             dicName);
3732                     }
3733                 case "wnn":
3734                     dicName = "WnnDic"
3735                     dicURL = WnnDic
3736                 case "sumomo":
3737                     dicName = "SumomoDic"
3738                     dicURL = SumomoDic
3739                 case "sijimi":
3740                     dicName = "SijimiDic"
3741                     dicURL = SijimiDic
3742                 case "jkl":
3743                     dicName = "JKLJaDic"
3744                     dicURL = JA_JKLDic
3745             }
3746             if debug {
3747                 fprintf(stderr,"--Id-- %v URL=%v\n\n",dicName,dicURL);
3748             }
3749             dicv := strings.Split(dicURL,",")
```

```
3750                    if debug {
3751                        fprintf(stderr,"--Id-- %v encoded data...\n",dicName)
3752                        fprintf(stderr,"Type: %v\n",dicv[0])
3753                        fprintf(stderr,"Body: %v\n",dicv[1])
3754                        fprintf(stderr,"\n")
3755                    }
3756                    body,_ := base64.StdEncoding.DecodeString(dicv[1])
3757                    dicBody = string(body)
3758                }
3759                if info {
3760                    fmt.Printf("--Id-- %v %v\n",dicName,dicURL)
3761                    fmt.Printf("%s\n",dicBody)
3762                }
3763                if debug {
3764                    fprintf(stderr,"--Id-- dicName %v text...\n",dicName)
3765                    fprintf(stderr,"%v\n",string(dicBody))
3766                }
3767                entv := strings.Split(dicBody,"\n");
3768                if info {
3769                    fprintf(stderr,"--Id-- %v scan...\n",dicName);
3770                }
3771                var added int = 0
3772                var dup int = 0
3773                for i,v := range entv {
3774                    var pat string
3775                    var out string
3776                    fmt.Sscanf(v,"%s %s",&pat,&out)
3777                    if len(pat) <= 0 {
3778                    }else{
3779                        if 0 <= isinDic(pat) {
3780                            dup += 1
3781                            continue
3782                        }
3783                        romkana[dicents] = RomKana{dicName,pat,out,0}
3784                        dicents += 1
3785                        added += 1
3786                        Romkan = append(Romkan,RomKana{dicName,pat,out,0})
3787                        if debug {
3788                            fmt.Printf("[%3v]:[%2v]%-8v [%2v]%v\n",
3789                                i,len(pat),pat,len(out),out)
3790                        }
3791                    }
3792                }
3793                if !silent {
3794                    url := dicURL
3795                    if strBegins(url,"data:") {
3796                        url = "builtin"
3797                    }
3798                    fprintf(stderr,"--Id-- %v scan... %v added, %v dup. / %v total (%v)\n",
3799                        dicName,added,dup,len(Romkan),url);
3800                }
3801                // should sort by pattern length for conclete match, for performance
3802                if debug {
3803                    arg = "" // search pattern
3804                    dump = true
3805                }
3806            }
3807            if cmd == DIC_COM_DUMP || dump {
3808                fprintf(stderr,"--Id-- %v dump... %v entries:\n",dicName,len(Romkan));
3809                var match = 0
3810                for i := 0; i < len(Romkan); i++ {
3811                    dic := Romkan[i].dic
3812                    pat := Romkan[i].pat
3813                    out := Romkan[i].out
3814                    if arg == "" || 0 <= strings.Index(pat,arg)||0 <= strings.Index(out,arg) {
3815                        fmt.Printf("\\\\%v\t%v [%2v]%-8v [%2v]%v\n",
3816                            i,dic,len(pat),pat,len(out),out)
3817                        match += 1
3818                    }
3819                }
3820                fprintf(stderr,"--Id-- %v matched %v / %v entries:\n",arg,match,len(Romkan));
3821            }
3822    }
3823    func loadDefaultDic(dic int){
3824        if( 0 < len(Romkan) ){
3825            return
3826        }
3827        //fprintf(stderr,"\r\n")
3828        xDic([]string{"dic",DIC_COM_LOAD});
3829
3830        var info = false
3831        if info {
3832            fprintf(stderr,"--Id-- Conguraturations!! WorldDic is now activated.\r\n")
3833            fprintf(stderr,"--Id-- enter \"dic\" command for help.\r\n")
3834        }
3835    }
3836    func readDic()(int){
3837        /*
3838        var rk *os.File;
3839        var dic = "MyIME-dic.txt";
3840        //rk = fopen("romkana.txt","r");
3841        //rk = fopen("JK-JA-morse-dic.txt","r");
3842        rk = fopen(dic,"r");
3843        if( rk == NULL_FP ){
3844            if( true ){
3845                fprintf(stderr,"--%s-- Could not load %s\n",MyIMEVER,dic);
3846            }
3847            return -1;
3848        }
3849        if( true ){
3850            var di int;
3851            var line = make(StrBuff,1024);
3852            var pat string
3853            var out string
3854            for di = 0; di < 1024; di++ {
3855                if( fgets(line,sizeof(line),rk) == NULLSP ){
3856                    break;
3857                }
3858                fmt.Sscanf(string(line[0:strlen(line)]),"%s %s",&pat,&out);
3859                //sscanf(line,"%s %[^\r\n]",&pat,&out);
3860                romkana[di].pat = pat;
3861                romkana[di].out = out;
3862                //fprintf(stderr,"--Dd- %-10s %s\n",pat,out)
3863            }
3864            dicents += di
3865            if( false ){
3866                fprintf(stderr,"--%s-- loaded romkana.txt [%d]\n",MyIMEVER,di);
3867                for di = 0; di < dicents; di++ {
3868                    fprintf(stderr,
3869                        "%s %s\n",romkana[di].pat,romkana[di].out);
3870                }
3871            }
3872        }
3873        fclose(rk);
3874
```

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

```
4000        if true {
4001            if EventRecvFd < 0 {
4002                var pv = []int{-1,-1}
4003                syscall.Pipe(pv)
4004                EventRecvFd = pv[0]
4005                EventSendFd = pv[1]
4006                //fmt.Printf("--De-- EventPipe created[%v,%v]\n",EventRecvFd,EventSendFd)
4007            }
4008        }else{
4009            if EventRecvFd < 0 {
4010                // the document differs from this spec
4011                // https://golang.org/src/syscall/syscall_unix.go?s=8096:8158#L340
4012                sv,err := syscall.Socketpair(syscall.AF_UNIX,syscall.SOCK_STREAM,0)
4013                EventRecvFd = sv[0]
4014                EventSendFd = sv[1]
4015                if err != nil {
4016                    fmt.Printf("--De-- EventSock created[%v,%v](%v)\n",
4017                        EventRecvFd,EventSendFd,err)
4018                }
4019            }
4020        }
4021        var buf = []byte{ byte(event)}
4022        n,err := syscall.Write(EventSendFd,buf)
4023        if err != nil {
4024            fmt.Printf("--De-- putEvent[%v](%3v)(%v %v)\n",EventSendFd,event,n,err)
4025        }
4026    }
4027    func ungets(str string){
4028        for _,ch := range str {
4029            putEvent(int(ch),0)
4030        }
4031    }
4032    func (gsh*GshContext)xReplay(argv[]string){
4033        hix := 0
4034        tempo := 1.0
4035        xtempo := 1.0
4036        repeat := 1
4037
4038        for _,a := range argv { // tempo
4039            if strBegins(a,"x") {
4040                fmt.Sscanf(a[1:],"%f",&xtempo)
4041                tempo = 1 / xtempo
4042                //fprintf(stderr,"--Dr-- tempo=[%v]%v\n",a[2:],tempo);
4043            }else
4044            if strBegins(a,"r") { // repeat
4045                fmt.Sscanf(a[1:],"%v",&repeat)
4046            }else
4047            if strBegins(a,"!") {
4048                fmt.Sscanf(a[1:],"%d",&hix)
4049            }else{
4050                fmt.Sscanf(a,"%d",&hix)
4051            }
4052        }
4053        if hix == 0 || len(argv) <= 1 {
4054            hix = len(gsh.CommandHistory)-1
4055        }
4056        fmt.Printf("--Ir-- Replay(!%v x%v r%v)\n",hix,xtempo,repeat)
4057        //dumpEvents(hix)
4058        //gsh.xScanReplay(hix,false,repeat,tempo,argv)
4059        go gsh.xScanReplay(hix,true,repeat,tempo,argv)
4060    }
4061
4062    // <a href="https://golang.org/pkg/syscall/#FdSet">syscall.Select</a>
4063    // 2020-0827 GShell-0.2.3
4064    func FpollIn1(fp *os.File,usec int)(uintptr){
4065        nfd := 1
4066
4067        rdv := syscall.FdSet {}
4068        fd1 := fp.Fd()
4069        bank1 := fd1/32
4070        mask1 := int32(1 << fd1)
4071        rdv.Bits[bank1] = mask1
4072
4073        fd2 := -1
4074        bank2 := -1
4075        var mask2 int32 = 0
4076
4077        if 0 <= EventRecvFd {
4078            fd2 = EventRecvFd
4079            nfd = fd2 + 1
4080            bank2 = fd2/32
4081            mask2 = int32(1 << fd2)
4082            rdv.Bits[bank2] |= mask2
4083            //fmt.Printf("--De-- EventPoll mask added [%d][%v][%v]\n",fd2,bank2,mask2)
4084        }
4085
4086        tout := syscall.NsecToTimeval(int64(usec*1000))
4087        //n,err := syscall.Select(nfd,&rdv,nil,nil,&tout) // spec. mismatch
4088        err := syscall.Select(nfd,&rdv,nil,nil,&tout)
4089        if err != nil {
4090            //fmt.Printf("--De-- select() err(%v)\n",err)
4091        }
4092        if err == nil {
4093            if 0 <= fd2 && (rdv.Bits[bank2] & mask2) != 0 {
4094                if false {
4095                    fmt.Printf("--De-- got Event\n")
4096                }
4097                return uintptr(EventFdOffset + fd2)
4098            }else
4099            if (rdv.Bits[bank1] & mask1) != 0 {
4100                return uintptr(NormalFdOffset + fd1)
4101            }else{
4102                return 1
4103            }
4104        }else{
4105            return 0
4106        }
4107    }
4108    func fgetcTimeout1(fp *os.File,usec int)(int){
4109      READ1:
4110        readyFd := FpollIn1(fp,usec)
4111        if readyFd < 100 {
4112            return EV_TIMEOUT
4113        }
4114
4115        var buf [1]byte
4116
4117        if EventFdOffset <= readyFd {
4118            fd := int(readyFd-EventFdOffset)
4119            _,err := syscall.Read(fd,buf[0:1])
4120            if( err != nil ){
4121                return EOF;
4122            }else{
4123                if buf[0] == EV_MODE {
4124                    recvEvent(fd)
```

```
4125                    goto READ1
4126            }
4127            return int(buf[0])
4128        }
4129    }
4130
4131    _,err := fp.Read(buf[0:1])
4132    if( err != nil ){
4133        return EOF;
4134    }else{
4135        return int(buf[0])
4136    }
4137 }
4138
4139 func visibleChar(ch int)(string){
4140    switch {
4141        case '!' <= ch && ch <= '~':
4142            return string(ch)
4143    }
4144    switch ch {
4145        case ' ': return "\\s"
4146        case '\n': return "\\n"
4147        case '\r': return "\\r"
4148        case '\t': return "\\t"
4149    }
4150    switch ch {
4151        case 0x00: return "NUL"
4152        case 0x07: return "BEL"
4153        case 0x08: return "BS"
4154        case 0x0E: return "SO"
4155        case 0x0F: return "SI"
4156        case 0x1B: return "ESC"
4157        case 0x7F: return "DEL"
4158    }
4159    switch ch {
4160        case EV_IDLE: return fmt.Sprintf("IDLE")
4161        case EV_MODE: return fmt.Sprintf("MODE")
4162    }
4163    return fmt.Sprintf("%X",ch)
4164 }
4165 func recvEvent(fd int){
4166    var buf = make([]byte,1)
4167    _,_ = syscall.Read(fd,buf[0:1])
4168    if( buf[0] != 0 ){
4169        romkanmode = true
4170    }else{
4171        romkanmode = false
4172    }
4173 }
4174 func (gsh*GshContext)xScanReplay(hix int,replay bool,repeat int,tempo float64,argv[]string){
4175    var Start time.Time
4176    var events = []Event{}
4177    for _,e := range Events {
4178        if hix == 0 || e.CmdIndex == hix {
4179            events = append(events,e)
4180        }
4181    }
4182    elen := len(events)
4183    if 0 < elen {
4184        if events[elen-1].event == EV_IDLE {
4185            events = events[0:elen-1]
4186        }
4187    }
4188    for r := 0; r < repeat; r++ {
4189        for i,e := range events {
4190            nano := e.when.Nanosecond()
4191            micro := nano / 1000
4192            if Start.Second() == 0 {
4193                Start = time.Now()
4194            }
4195            diff := time.Now().Sub(Start)
4196            if replay {
4197                if e.event != EV_IDLE {
4198                    putEvent(e.event,0)
4199                    if e.event == EV_MODE { // event with arg
4200                        putEvent(int(e.evarg),0)
4201                    }
4202                }
4203            }else{
4204                fmt.Printf("%7.3fms #%-3v !%-3v [%v.%06d] %3v %02X %-4v %10.3fms\n",
4205                    float64(diff)/1000000.0,
4206                    i,
4207                    e.CmdIndex,
4208                    e.when.Format(time.Stamp),micro,
4209                    e.event,e.event,visibleChar(e.event),
4210                    float64(e.evarg)/1000000.0)
4211            }
4212            if e.event == EV_IDLE {
4213                d := time.Duration(float64(time.Duration(e.evarg)) * tempo)
4214                //nsleep(time.Duration(e.evarg))
4215                nsleep(d)
4216            }
4217        }
4218    }
4219 }
4220 func dumpEvents(arg[]string){
4221    hix := 0
4222    if 1 < len(arg) {
4223        fmt.Sscanf(arg[1],"%d",&hix)
4224    }
4225    for i,e := range Events {
4226        nano := e.when.Nanosecond()
4227        micro := nano / 1000
4228        //if e.event != EV_TIMEOUT {
4229        if hix == 0 || e.CmdIndex == hix {
4230            fmt.Printf("#%-3v !%-3v [%v.%06d] %3v %02X %-4v %10.3fms\n",i,
4231                e.CmdIndex,
4232                e.when.Format(time.Stamp),micro,
4233                e.event,e.event,visibleChar(e.event),float64(e.evarg)/1000000.0)
4234        }
4235        //}
4236    }
4237 }
4238 func fgetcTimeout(fp *os.File,usec int)(int){
4239    ch := fgetcTimeout1(fp,usec)
4240    if ch != EV_TIMEOUT {
4241        now := time.Now()
4242        if 0 < len(Events) {
4243            last := Events[len(Events)-1]
4244            dura := int64(now.Sub(last.when))
4245            Events = append(Events,Event{last.when,EV_IDLE,dura,last.CmdIndex})
4246        }
4247        Events = append(Events,Event{time.Now(),ch,0,CmdIndex})
4248    }
4249    return ch
```

```
4250 }
4251
4252 var TtyMaxCol = 72 // to be obtained by ioctl?
4253 var EscTimeout = (100*1000)
4254 var (
4255     MODE_VicMode    bool    // vi compatible command mode
4256     MODE_ShowMode   bool
4257     romkanmode  bool    // shown translation mode, the mode to be retained
4258     MODE_Recursive  bool    // recursive translation
4259     MODE_CapsLock   bool    // software CapsLock
4260     MODE_LowerLock  bool    // force lower-case character lock
4261     MODE_ViInsert   int // visible insert mode, should be like "I" icon in X Window
4262     MODE_ViTrace    bool    // output newline before translation
4263 )
4264 type IInput struct {
4265     lno     int
4266     lastlno     int
4267     pch     []int   // input queue
4268     prompt      string
4269     line        string
4270     right       string
4271     inJmode     bool
4272     pinJmode    bool
4273     waitingMeta string  // waiting meta character
4274     LastCmd     string
4275 }
4276 func (iin*IInput)Getc(timeoutUs int)(int){
4277     ch1 := EOF
4278     ch2 := EOF
4279     ch3 := EOF
4280     if( 0 < len(iin.pch) ){ // deQ
4281         ch1 = iin.pch[0]
4282         iin.pch = iin.pch[1:]
4283     }else{
4284         ch1 = fgetcTimeout(stdin,timeoutUs);
4285     }
4286     if( ch1 == 033 ){ /// escape sequence
4287         ch2 = fgetcTimeout(stdin,EscTimeout);
4288         if( ch2 == EV_TIMEOUT ){
4289         }else{
4290             ch3 = fgetcTimeout(stdin,EscTimeout);
4291             if( ch3 == EV_TIMEOUT ){
4292                 iin.pch = append(iin.pch,ch2) // enQ
4293             }else{
4294                 switch( ch2 ){
4295                     default:
4296                         iin.pch = append(iin.pch,ch2) // enQ
4297                         iin.pch = append(iin.pch,ch3) // enQ
4298                     case '[':
4299                         switch( ch3 ){
4300                             case 'A': ch1 = GO_UP; // ^
4301                             case 'B': ch1 = GO_DOWN; // v
4302                             case 'C': ch1 = GO_RIGHT; // >
4303                             case 'D': ch1 = GO_LEFT; // <
4304                             case '3':
4305                         ch4 := fgetcTimeout(stdin,EscTimeout);
4306                                 if( ch4 == '~' ){
4307 //fprintf(stderr,"x[%02X %02X %02X %02X]\n",ch1,ch2,ch3,ch4);
4308                                     ch1 = DEL_RIGHT
4309                                 }
4310                         }
4311                     case '\\':
4312                     //ch4 := fgetcTimeout(stdin,EscTimeout);
4313                     //fprintf(stderr,"y[%02X %02X %02X %02X]\n",ch1,ch2,ch3,ch4);
4314                         switch( ch3 ){
4315                             case '~': ch1 = DEL_RIGHT
4316                         }
4317                 }
4318             }
4319         }
4320     }
4321     return ch1
4322 }
4323 func (inn*IInput)clearline(){
4324     var i int
4325     fprintf(stderr,"\r");
4326     // should be ANSI ESC sequence
4327     for i = 0; i < TtyMaxCol; i++ { // to the max. position in this input action
4328         fputc(' ',os.Stderr);
4329     }
4330     fprintf(stderr,"\r");
4331 }
4332 func (iin*IInput)Redraw(){
4333     redraw(iin,iin.lno,iin.line,iin.right)
4334 }
4335 func redraw(iin *IInput,lno int,line string,right string){
4336     inMeta := false
4337     showMode := ""
4338     showMeta := "" // visible Meta mode on the cursor position
4339     showLino := fmt.Sprintf("!%d! ",lno)
4340     InsertMark := "" // in visible insert mode
4341
4342     if MODE_VicMode {
4343     }else
4344     if 0 < len(iin.right) {
4345         InsertMark = " "
4346     }
4347
4348     if( 0 < len(iin.waitingMeta) ){
4349         inMeta = true
4350         if iin.waitingMeta[0] != 033 {
4351             showMeta = iin.waitingMeta
4352         }
4353     }
4354     if( romkanmode ){
4355         //romkanmark = " *";
4356     }else{
4357         //romkanmark = "";
4358     }
4359     if MODE_ShowMode {
4360         romkan := "--"
4361         inmeta := "-"
4362         inveri := ""
4363         if MODE_CapsLock {
4364             inmeta = "A"
4365         }
4366         if MODE_LowerLock {
4367             inmeta = "a"
4368         }
4369         if MODE_ViTrace {
4370             inveri = "v"
4371         }
4372         if MODE_VicMode {
4373             inveri = ":"
4374         }
```

```
4375            if romkanmode {
4376                romkan = "\343\201\202"
4377                if MODE_CapsLock {
4378                    inmeta = "R"
4379                }else{
4380                    inmeta = "r"
4381                }
4382            }
4383            if inMeta {
4384                inmeta = "\\"
4385            }
4386            showMode = "["+romkan+inmeta+inveri+"]";
4387        }
4388        Pre := "\r" + showMode + showLino
4389        Output := ""
4390        Left := ""
4391        Right := ""
4392        if romkanmode {
4393            Left = convs(line)
4394            Right = InsertMark+convs(right)
4395        }else{
4396            Left = line
4397            Right = InsertMark+right
4398        }
4399        Output = Pre+Left
4400        if MODE_ViTrace {
4401            Output += iin.LastCmd
4402        }
4403        Output += showMeta+Right
4404        for len(Output) < TtyMaxCol { // to the max. position that may be dirty
4405            Output += " "
4406            // should be ANSI ESC sequence
4407            // not necessary just after newline
4408        }
4409        Output += Pre+Left+showMeta // to set the cursor to the current input position
4410        fprintf(stderr,"%s",Output)
4411
4412        if MODE_ViTrace {
4413            if 0 < len(iin.LastCmd) {
4414                iin.LastCmd = ""
4415                fprintf(stderr,"\r\n")
4416            }
4417        }
4418 }
4419 // <a href="https://golang.org/pkg/unicode/utf8/">utf8</a>
4420 func delHeadChar(str string)(rline string,head string){
4421        _,clen := utf8.DecodeRune([]byte(str))
4422        head = string(str[0:clen])
4423        return str[clen:],head
4424 }
4425 func delTailChar(str string)(rline string, last string){
4426        var i = 0
4427        var clen = 0
4428        for {
4429            _,siz := utf8.DecodeRune([]byte(str)[i:])
4430            if siz <= 0 { break }
4431            clen = siz
4432            i += siz
4433        }
4434        last = str[len(str)-clen:]
4435        return str[0:len(str)-clen],last
4436 }
4437
4438 // 3> for output and history
4439 // 4> for keylog?
4440 // <a name="getline">Command Line Editor</a>
4441 func xgetline(lno int, prevline string, gsh*GshContext)(string){
4442        var iin IInput
4443        iin.lastlno = lno
4444        iin.lno = lno
4445
4446        CmdIndex = len(gsh.CommandHistory)
4447        if( isatty(0) == 0 ){
4448            if( sfgets(&iin.line,LINESIZE,stdin) == NULL ){
4449                iin.line = "exit\n";
4450            }else{
4451            }
4452            return iin.line
4453        }
4454        if( true ){
4455            //var pts string;
4456            //pts = ptsname(0);
4457            //pts = ttyname(0);
4458            //fprintf(stderr,"--pts[0] = %s\n",pts?pts:"?");
4459        }
4460        if( false ){
4461            fprintf(stderr,"! ");
4462            fflush(stderr);
4463            sfgets(&iin.line,LINESIZE,stdin);
4464            return iin.line
4465        }
4466        system("/bin/stty -echo -icanon");
4467        xline := iin.xgetline1(prevline,gsh)
4468        system("/bin/stty echo sane");
4469        return xline
4470 }
4471 func (iin*IInput)Translate(cmdch int){
4472        romkanmode = !romkanmode;
4473        if MODE_ViTrace {
4474            fprintf(stderr,"%v\r\n",string(cmdch));
4475        }else
4476        if( cmdch == 'J' ){
4477            fprintf(stderr,"J\r\n");
4478            iin.inJmode = true
4479        }
4480        iin.Redraw();
4481        loadDefaultDic(cmdch);
4482        iin.Redraw();
4483 }
4484 func (iin*IInput)Replace(cmdch int){
4485        iin.LastCmd = fmt.Sprintf("\\%v",string(cmdch))
4486        iin.Redraw();
4487        loadDefaultDic(cmdch);
4488        dst := convs(iin.line+iin.right);
4489        iin.line = dst
4490        iin.right = ""
4491        if( cmdch == 'I' ){
4492            fprintf(stderr,"I\r\n");
4493            iin.inJmode = true
4494        }
4495        iin.Redraw();
4496 }
4497 // aa 12 a1a1
4498 func isAlpha(ch rune)(bool){
4499        if 'a' <= ch && ch <= 'z' || 'A' <= ch && ch <= 'Z' {
```

```
4500            return true
4501        }
4502        return false
4503 }
4504 func isAlnum(ch rune)(bool){
4505        if 'a' <= ch && ch <= 'z' || 'A' <= ch && ch <= 'Z' {
4506            return true
4507        }
4508        if '0' <= ch && ch <= '9' {
4509            return true
4510        }
4511        return false
4512 }
4513
4514 // 0.2.8 2020-0901 created
4515 // <a href="https://golang.org/pkg/unicode/utf8/#DecodeRuneInString">DecodeRuneInString</a>
4516 func (iin*IInput)GotoTOPW(){
4517        str := iin.line
4518        i := len(str)
4519        if i <= 0 {
4520            return
4521        }
4522        //i0 := i
4523        i -= 1
4524        lastSize := 0
4525        var lastRune rune
4526        var found = -1
4527        for 0 < i { // skip preamble spaces
4528            lastRune,lastSize = utf8.DecodeRuneInString(str[i:])
4529            if !isAlnum(lastRune) { // character, type, or string to be searched
4530                i -= lastSize
4531                continue
4532            }
4533            break
4534        }
4535        for 0 < i {
4536            lastRune,lastSize = utf8.DecodeRuneInString(str[i:])
4537            if lastSize <= 0 { continue } // not the character top
4538            if !isAlnum(lastRune) { // character, type, or string to be searched
4539                found = i
4540                break
4541            }
4542            i -= lastSize
4543        }
4544        if found < 0 && i == 0 {
4545            found = 0
4546        }
4547        if 0 <= found {
4548            if isAlnum(lastRune) { // or non-kana character
4549            }else{ // when positioning to the top o the word
4550                i += lastSize
4551            }
4552            iin.right = str[i:] + iin.right
4553            if 0 < i {
4554                iin.line = str[0:i]
4555            }else{
4556                iin.line = ""
4557            }
4558        }
4559        //fmt.Printf("\n(%d,%d,%d)[%s][%s]\n",i0,i,found,iin.line,iin.right)
4560        //fmt.Printf("") // set debug messae at the end of line
4561 }
4562 // 0.2.8 2020-0901 created
4563 func (iin*IInput)GotoENDW(){
4564        str := iin.right
4565        if len(str) <= 0 {
4566            return
4567        }
4568        lastSize := 0
4569        var lastRune rune
4570        var lastW = 0
4571        i := 0
4572        inWord := false
4573
4574        lastRune,lastSize = utf8.DecodeRuneInString(str[0:])
4575        if isAlnum(lastRune) {
4576            r,z := utf8.DecodeRuneInString(str[lastSize:])
4577            if 0 < z && isAlnum(r) {
4578                inWord = true
4579            }
4580        }
4581        for i < len(str) {
4582            lastRune,lastSize = utf8.DecodeRuneInString(str[i:])
4583            if lastSize <= 0 { break } // broken data?
4584            if !isAlnum(lastRune) { // character, type, or string to be searched
4585                break
4586            }
4587            lastW = i // the last alnum if in alnum word
4588            i += lastSize
4589        }
4590        if inWord {
4591            goto DISP
4592        }
4593        for i < len(str) {
4594            lastRune,lastSize = utf8.DecodeRuneInString(str[i:])
4595            if lastSize <= 0 { break } // broken data?
4596            if isAlnum(lastRune) { // character, type, or string to be searched
4597                break
4598            }
4599            i += lastSize
4600        }
4601        for i < len(str) {
4602            lastRune,lastSize = utf8.DecodeRuneInString(str[i:])
4603            if lastSize <= 0 { break } // broken data?
4604            if !isAlnum(lastRune) { // character, type, or string to be searched
4605                break
4606            }
4607            lastW = i
4608            i += lastSize
4609        }
4610 DISP:
4611        if 0 < lastW {
4612            iin.line = iin.line + str[0:lastW]
4613            iin.right = str[lastW:]
4614        }
4615        //fmt.Printf("\n(%d)[%s][%s]\n",i,iin.line,iin.right)
4616        //fmt.Printf("") // set debug messae at the end of line
4617 }
4618 // 0.2.8 2020-0901 created
4619 func (iin*IInput)GotoNEXTW(){
4620        str := iin.right
4621        if len(str) <= 0 {
4622            return
4623        }
4624        lastSize := 0
```

```
4625          var lastRune rune
4626          var found = -1
4627          i := 1
4628          for i < len(str) {
4629              lastRune,lastSize = utf8.DecodeRuneInString(str[i:])
4630              if lastSize <= 0 { break } // broken data?
4631              if !isAlnum(lastRune) { // character, type, or string to be searched
4632                  found = i
4633                  break
4634              }
4635              i += lastSize
4636          }
4637          if 0 < found {
4638              if isAlnum(lastRune) { // or non-kana character
4639              }else{ // when positioning to the top o the word
4640                  found += lastSize
4641              }
4642              iin.line = iin.line + str[0:found]
4643              if 0 < found {
4644                  iin.right = str[found:]
4645              }else{
4646                  iin.right = ""
4647              }
4648          }
4649          //fmt.Printf("\n(%d)[%s][%s]\n",i,iin.line,iin.right)
4650          //fmt.Printf("") // set debug messae at the end of line
4651  }
4652  // 0.2.8 2020-0902 created
4653  func (iin*IInput)GotoPAIRCH(){
4654          str := iin.right
4655          if len(str) <= 0 {
4656              return
4657          }
4658          lastRune,lastSize := utf8.DecodeRuneInString(str[0:])
4659          if lastSize <= 0 {
4660              return
4661          }
4662          forw := false
4663          back := false
4664          pair := ""
4665          switch string(lastRune){
4666              case "{": pair = "}"; forw = true
4667              case "}": pair = "{"; back = true
4668              case "(": pair = ")"; forw = true
4669              case ")": pair = "("; back = true
4670              case "[": pair = "]"; forw = true
4671              case "]": pair = "["; back = true
4672              case "<": pair = ">"; forw = true
4673              case ">": pair = "<"; back = true
4674              case "\"": pair = "\""; // context depednet, can be f" or back-double quote
4675              case "'": pair = "'"; // context depednet, can be f' or back-quote
4676              // case Japanese Kakkos
4677          }
4678          if forw {
4679              iin.SearchForward(pair)
4680          }
4681          if back {
4682              iin.SearchBackward(pair)
4683          }
4684  }
4685  // 0.2.8 2020-0902 created
4686  func (iin*IInput)SearchForward(pat string)(bool){
4687          right := iin.right
4688          found := -1
4689          i := 0
4690          if strBegins(right,pat) {
4691              _,z := utf8.DecodeRuneInString(right[i:])
4692              if 0 < z {
4693                  i += z
4694              }
4695          }
4696          for i < len(right) {
4697              if strBegins(right[i:],pat) {
4698                  found = i
4699                  break
4700              }
4701              _,z := utf8.DecodeRuneInString(right[i:])
4702              if z <= 0 { break }
4703              i += z
4704          }
4705          if 0 <= found {
4706              iin.line = iin.line + right[0:found]
4707              iin.right = iin.right[found:]
4708              return true
4709          }else{
4710              return false
4711          }
4712  }
4713  // 0.2.8 2020-0902 created
4714  func (iin*IInput)SearchBackward(pat string)(bool){
4715          line := iin.line
4716          found := -1
4717          i := len(line)-1
4718          for i = i; 0 <= i; i-- {
4719              _,z := utf8.DecodeRuneInString(line[i:])
4720              if z <= 0 {
4721                  continue
4722              }
4723              //fprintf(stderr,"-- %v %v\n",pat,line[i:])
4724              if strBegins(line[i:],pat) {
4725                  found = i
4726                  break
4727              }
4728          }
4729          //fprintf(stderr,"--%d\n",found)
4730          if 0 <= found {
4731              iin.right = line[found:] + iin.right
4732              iin.line = line[0:found]
4733              return true
4734          }else{
4735              return false
4736          }
4737  }
4738  // 0.2.8 2020-0902 created
4739  // search from top, end, or current position
4740  func (gsh*GshContext)SearchHistory(pat string, forw bool)(bool,string){
4741          if forw {
4742              for _,v := range gsh.CommandHistory {
4743                  if 0 <= strings.Index(v.CmdLine,pat) {
4744                      //fprintf(stderr,"\n--De-- found !%v [%v]%v\n",i,pat,v.CmdLine)
4745                      return true,v.CmdLine
4746                  }
4747              }
4748          }else{
4749              hlen := len(gsh.CommandHistory)
```

```
4750            for i := hlen-1; 0 < i ; i-- {
4751                v := gsh.CommandHistory[i]
4752                if 0 <= strings.Index(v.CmdLine,pat) {
4753                    //fprintf(stderr,"\n--De-- found !%v [%v]%v\n",i,pat,v.CmdLine)
4754                    return f true,v.CmdLine
4755                }
4756            }
4757        }
4758        //fprintf(stderr,"\n--De-- not-found(%v)\n",pat)
4759        return false,"(Not Found in History)"
4760 }
4761 // 0.2.8 2020-0902 created
4762 func (iin*IInput)GotoFORWSTR(pat string,gsh*GshContext){
4763        found := false
4764        if 0 < len(iin.right) {
4765            found = iin.SearchForward(pat)
4766        }
4767        if !found {
4768            found,line := gsh.SearchHistory(pat,true)
4769            if found {
4770                iin.line = line
4771                iin.right = ""
4772            }
4773        }
4774 }
4775 func (iin*IInput)GotoBACKSTR(pat string, gsh*GshContext){
4776        found := false
4777        if 0 < len(iin.line) {
4778            found = iin.SearchBackward(pat)
4779        }
4780        if !found {
4781            found,line := gsh.SearchHistory(pat,false)
4782            if found {
4783                iin.line = line
4784                iin.right = ""
4785            }
4786        }
4787 }
4788 func (iin*IInput)getstring1(prompt string)(string){ // should be editable
4789        iin.clearline();
4790        fprintf(stderr,"\r%v",prompt)
4791        str := ""
4792        for {
4793            ch := iin.Getc(10*1000*1000)
4794            if ch == '\n' || ch == '\r' {
4795                break
4796            }
4797            sch := string(ch)
4798            str += sch
4799            fprintf(stderr,"%s",sch)
4800        }
4801        return str
4802 }
4803
4804 // search pattern must be an array and selectable with ^N/^P
4805 var SearchPat = ""
4806 var SearchForw = true
4807
4808 func (iin*IInput)xgetline1(prevline string, gsh*GshContext)(string){
4809        var ch int;
4810
4811        MODE_ShowMode = false
4812        MODE_VicMode = false
4813        iin.Redraw();
4814        first := true
4815
4816        for cix := 0; ; cix++ {
4817            iin.pinJmode = iin.inJmode
4818            iin.inJmode = false
4819
4820            ch = iin.Getc(1000*1000)
4821
4822            if ch != EV_TIMEOUT && first {
4823                first = false
4824                mode := 0
4825                if romkanmode {
4826                    mode = 1
4827                }
4828                now := time.Now()
4829                Events = append(Events,Event{now,EV_MODE,int64(mode),CmdIndex})
4830            }
4831            if ch == 033 {
4832                MODE_ShowMode = true
4833                MODE_VicMode = !MODE_VicMode
4834                iin.Redraw();
4835                continue
4836            }
4837            if MODE_VicMode {
4838                switch ch {
4839                    case '0': ch = GO_TOPL
4840                    case '$': ch = GO_ENDL
4841                    case 'b': ch = GO_TOPW
4842                    case 'e': ch = GO_ENDW
4843                    case 'w': ch = GO_NEXTW
4844                    case '%': ch = GO_PAIRCH
4845
4846                    case 'j': ch = GO_DOWN
4847                    case 'k': ch = GO_UP
4848                    case 'h': ch = GO_LEFT
4849                    case 'l': ch = GO_RIGHT
4850                    case 'x': ch = DEL_RIGHT
4851                    case 'a': MODE_VicMode = !MODE_VicMode
4852                        ch = GO_RIGHT
4853                    case 'i': MODE_VicMode = !MODE_VicMode
4854                        iin.Redraw();
4855                        continue
4856                    case '~':
4857                        right,head := delHeadChar(iin.right)
4858                        if len([]byte(head)) == 1 {
4859                            ch = int(head[0])
4860                            if( 'a' <= ch && ch <= 'z' ){
4861                                ch = ch + 'A'-'a'
4862                            }else
4863                            if( 'A' <= ch && ch <= 'Z' ){
4864                                ch = ch + 'a'-'A'
4865                            }
4866                            iin.right = string(ch) + right
4867                        }
4868                        iin.Redraw();
4869                        continue
4870                    case 'f': // GO_FORWCH
4871                        iin.Redraw();
4872                        ch = iin.Getc(3*1000*1000)
4873                        if ch == EV_TIMEOUT {
4874                            iin.Redraw();
```

```
4875                        continue
4876                    }
4877                    SearchPat = string(ch)
4878                    SearchForw = true
4879                    iin.GotoFORWSTR(SearchPat,gsh)
4880                    iin.Redraw();
4881                    continue
4882                case '/':
4883                    SearchPat = iin.getstring1("/") // should be editable
4884                    SearchForw = true
4885                    iin.GotoFORWSTR(SearchPat,gsh)
4886                    iin.Redraw();
4887                    continue
4888                case '?':
4889                    SearchPat = iin.getstring1("?") // should be editable
4890                    SearchForw = false
4891                    iin.GotoBACKSTR(SearchPat,gsh)
4892                    iin.Redraw();
4893                    continue
4894                case 'n':
4895                    if SearchForw {
4896                        iin.GotoFORWSTR(SearchPat,gsh)
4897                    }else{
4898                        iin.GotoBACKSTR(SearchPat,gsh)
4899                    }
4900                    iin.Redraw();
4901                    continue
4902                case 'N':
4903                    if !SearchForw {
4904                        iin.GotoFORWSTR(SearchPat,gsh)
4905                    }else{
4906                        iin.GotoBACKSTR(SearchPat,gsh)
4907                    }
4908                    iin.Redraw();
4909                    continue
4910            }
4911        }
4912        switch ch {
4913            case GO_TOPW:
4914                iin.GotoTOPW()
4915                iin.Redraw();
4916                continue
4917            case GO_ENDW:
4918                iin.GotoENDW()
4919                iin.Redraw();
4920                continue
4921            case GO_NEXTW:
4922                // to next space then
4923                iin.GotoNEXTW()
4924                iin.Redraw();
4925                continue
4926            case GO_PAIRCH:
4927                iin.GotoPAIRCH()
4928                iin.Redraw();
4929                continue
4930        }
4931
4932        //fprintf(stderr,"A[%02X]\n",ch);
4933        if( ch == '\\' || ch == 033 ){
4934            MODE_ShowMode = true
4935            metach := ch
4936            iin.waitingMeta = string(ch)
4937            iin.Redraw();
4938                // set cursor //fprintf(stderr,"???\b\b\b")
4939            ch = fgetcTimeout(stdin,2000*1000)
4940                // reset cursor
4941            iin.waitingMeta = ""
4942
4943            cmdch := ch
4944            if( ch == EV_TIMEOUT ){
4945                if metach == 033 {
4946                    continue
4947                }
4948                ch = metach
4949            }else
4950            /*
4951            if( ch == 'm' || ch == 'M' ){
4952                mch := fgetcTimeout(stdin,1000*1000)
4953                if mch == 'r' {
4954                    romkanmode = true
4955                }else{
4956                    romkanmode = false
4957                }
4958                continue
4959            }else
4960            */
4961            if( ch == 'k' || ch == 'K' ){
4962                MODE_Recursive = !MODE_Recursive
4963                iin.Translate(cmdch);
4964                continue
4965            }else
4966            if( ch == 'j' || ch == 'J' ){
4967                iin.Translate(cmdch);
4968                continue
4969            }else
4970            if( ch == 'i' || ch == 'I' ){
4971                iin.Replace(cmdch);
4972                continue
4973            }else
4974            if( ch == 'l' || ch == 'L' ){
4975                MODE_LowerLock = !MODE_LowerLock
4976                MODE_CapsLock = false
4977                if MODE_ViTrace {
4978                    fprintf(stderr,"%v\r\n",string(cmdch));
4979                }
4980                iin.Redraw();
4981                continue
4982            }else
4983            if( ch == 'u' || ch == 'U' ){
4984                MODE_CapsLock = !MODE_CapsLock
4985                MODE_LowerLock = false
4986                if MODE_ViTrace {
4987                    fprintf(stderr,"%v\r\n",string(cmdch));
4988                }
4989                iin.Redraw();
4990                continue
4991            }else
4992            if( ch == 'v' || ch == 'V' ){
4993                MODE_ViTrace = !MODE_ViTrace
4994                if MODE_ViTrace {
4995                    fprintf(stderr,"%v\r\n",string(cmdch));
4996                }
4997                iin.Redraw();
4998                continue
4999            }else
```

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

```
5125                       iin.clearline();
5126                       iin.Redraw();
5127                       continue;
5128                   case DEL_RIGHT:
5129                       if( 0 < len(iin.right) ){
5130                           iin.right,_ = delHeadChar(iin.right)
5131                           iin.Redraw();
5132                       }
5133                       continue;
5134                   case 0x7F: // BS? not DEL
5135                       if( 0 < len(iin.line) ){
5136                           iin.line,_ = delTailChar(iin.line)
5137                           iin.Redraw();
5138                       }
5139                       /*
5140                       else
5141                       if( 0 < len(iin.right) ){
5142                           iin.right,_ = delHeadChar(iin.right)
5143                           iin.Redraw();
5144                       }
5145                       */
5146                       continue;
5147                   case 'H'-0x40:
5148                       if( 0 < len(iin.line) ){
5149                           iin.line,_ = delTailChar(iin.line)
5150                           iin.Redraw();
5151                       }
5152                       continue;
5153               }
5154               if( ch == '\n' || ch == '\r' ){
5155                   iin.line += iin.right;
5156                   iin.right = ""
5157                   iin.Redraw();
5158                   fputc(ch,stderr);
5159                   break;
5160               }
5161               if MODE_CapsLock {
5162                   if 'a' <= ch && ch <= 'z' {
5163                       ch = ch+'A'-'a'
5164                   }
5165               }
5166               if MODE_LowerLock {
5167                   if 'A' <= ch && ch <= 'Z' {
5168                       ch = ch+'a'-'A'
5169                   }
5170               }
5171               iin.line += string(ch);
5172               iin.Redraw();
5173       }
5174  EXIT:
5175       return iin.line + iin.right;
5176  }
5177
5178  func getline_main(){
5179       line := xgetline(0,"",nil)
5180       fprintf(stderr,"%s\n",line);
5181  /*
5182       dp = strpbrk(line,"\r\n");
5183       if( dp != NULL ){
5184           *dp = 0;
5185       }
5186
5187       if( 0 ){
5188           fprintf(stderr,"\n(%d)\n",int(strlen(line)));
5189       }
5190       if( lseek(3,0,0) == 0 ){
5191           if( romkanmode ){
5192               var buf [8*1024]byte;
5193               convs(line,buff);
5194               strcpy(line,buff);
5195           }
5196           write(3,line,strlen(line));
5197           ftruncate(3,lseek(3,0,SEEK_CUR));
5198           //fprintf(stderr,"outsize=%d\n",(int)lseek(3,0,SEEK_END));
5199           lseek(3,0,SEEK_SET);
5200           close(3);
5201       }else{
5202           fprintf(stderr,"\r\ngotline: ");
5203           trans(line);
5204           //printf("%s\n",line);
5205           printf("\n");
5206       }
5207  */
5208  }
5209  //== end ========================================================= getline
5210
5211  //
5212  // $USERHOME/.gsh/
5213  //       gsh-rc.txt, or gsh-configure.txt
5214  //            gsh-history.txt
5215  //            gsh-aliases.txt // should be conditional?
5216  //
5217  func (gshCtx *GshContext)gshSetupHomedir()(bool) {
5218       homedir,found := userHomeDir()
5219       if !found {
5220           fmt.Printf("--E-- You have no UserHomeDir\n")
5221           return true
5222       }
5223       gshhome := homedir + "/" + GSH_HOME
5224       _, err2 := os.Stat(gshhome)
5225       if err2 != nil {
5226           err3 := os.Mkdir(gshhome,0700)
5227           if err3 != nil {
5228               fmt.Printf("--E-- Could not Create %s (%s)\n",
5229                   gshhome,err3)
5230               return true
5231           }
5232           fmt.Printf("--I-- Created %s\n",gshhome)
5233       }
5234       gshCtx.GshHomeDir = gshhome
5235       return false
5236  }
5237  func setupGshContext()(GshContext,bool){
5238       gshPA := syscall.ProcAttr {
5239           "", // the staring directory
5240           os.Environ(), // environ[]
5241           []uintptr{os.Stdin.Fd(),os.Stdout.Fd(),os.Stderr.Fd()},
5242           nil, // OS specific
5243       }
5244       cwd, _ := os.Getwd()
5245       gshCtx := GshContext {
5246           cwd, // StartDir
5247           "", // GetLine
5248           []GChdirHistory { {cwd,time.Now(),0} }, // ChdirHistory
5249           gshPA,
```

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

```
5375            prevline = gline;
5376            hix++;
5377        }
5378        return *gshCtx
5379 }
5380 func main() {
5381     gshCtxBuf := GshContext{}
5382     gsh := &gshCtxBuf
5383     argv := os.Args
5384     if 1 < len(argv) {
5385         if isin("version",argv){
5386             gsh.showVersion(argv)
5387             return
5388         }
5389         comx := isinX("-c",argv)
5390         if 0 < comx {
5391             gshCtxBuf,err := setupGshContext()
5392             gsh := &gshCtxBuf
5393             if !err {
5394                 gsh.gshellv(argv[comx+1:])
5395             }
5396             return
5397         }
5398     }
5399     if 1 < len(argv) && isin("-s",argv) {
5400     }else{
5401         gsh.showVersion(append(argv,[]string{"-l","-a"}...))
5402     }
5403     script(nil)
5404     //gshCtx := script(nil)
5405     //gshell(gshCtx,"time")
5406 }
5407 //</div></details>
5408 //<details id="gsh-todo"><summary>Considerations</summary><div class="gsh-src">
5409 // - inter gsh communication, possibly running in remote hosts -- to be remote shell
5410 // - merged histories of multiple parallel gsh sessions
5411 // - alias as a function or macro
5412 // - instant alias end environ export to the permanent > ~/.gsh/gsh-alias and gsh-environ
5413 // - retrieval PATH of files by its type
5414 // - gsh as an IME with completion using history and file names as dictionaies
5415 // - gsh a scheduler in precise time of within a millisecond
5416 // - all commands have its subucomand after "---" symbol
5417 // - filename expansion by "-find" command
5418 // - history of ext code and output of each commoand
5419 // - "script" output for each command by pty-tee or telnet-tee
5420 // - $BUILTIN command in PATH to show the priority
5421 // - "?" symbol in the command (not as in arguments) shows help request
5422 // - searching command with wild card like: which ssh-*
5423 // - longformat prompt after long idle time (should dismiss by BS)
5424 // - customizing by building plugin and dynamically linking it
5425 // - generating syntactic element like "if" by macro expansion (like CPP) >> alias
5426 // - "!" symbol should be used for negation, don't wast it just for job control
5427 // - don't put too long output to tty, record it into GSH_HOME/session-id/comand-id.log
5428 // - making canonical form of command at the start adding quatation or white spaces
5429 // - name(a,b,c) ... use "(" and ")" to show both delimiter and realm
5430 // - name? or name! might be useful
5431 // - htar format - packing directory contents into a single html file using data scheme
5432 // - filepath substitution shold be done by each command, expecially in case of builtins
5433 // - @N substition for the history of working directory, and @spec for more generic ones
5434 // - @dir prefix to do the command at there, that means like (chdir @dir; command)
5435 // - GSH_PATH for plugins
5436 // - standard command output: list of data with name, size, resouce usage, modified time
5437 // - generic sort key option -nm name, -sz size, -ru rusage, -ts start-time, -tm mod-time
5438 //   -wc word-count, grep match line count, ...
5439 // - standard command execution result: a list of string, -tm, -ts, -ru, -sz, ...
5440 // - -tailf-filename like tail -f filename, repeat close and open before read
5441 // - max. size and max. duration and timeout of (generated) data transfer
5442 // - auto. numbering, aliasing, IME completion of file name (especially rm of quieer name)
5443 // - IME "?" at the top of the command line means searching history
5444 // - IME %d/0x10000/ %x/ffff/
5445 // - IME ESC to go the edit mode like in vi, and use :command as :s/x/y/g to edit history
5446 // - gsh in WebAssembly
5447 // - gsh as a HTTP server of online-manual
5448 //---END--- (^-^)/ITS more</div></details>
5449
5450 //<span class="gsh-golang-data">
5451
5452 var WorldDic = //<span id="gsh-world-dic">
5453 "data:text/dic;base64,"+
5454 "Ly8gTXlJTUUvvMC4wLjEg6L6e5pu4ICgyMDIwLTA4MTlhKQpzZWthaSdkuJbnlYwKa28g44GT"+
5455 "Cm5uIOOOCkwpuaSDjgasKY2hpIOOBoQp0aSDjgaEKaGEg44GvCnNlIOOBmwprYSDjgYsKaSDj"+
5456 "gYQK";
5457 //</span>
5458
5459 var WnnDic = //<span id="gsh-wnn-dic">
5460 "data:text/dic;base64,"+
5461 "PG1ldGEgY2hhcnNldD0iVVRGGLTgiPgo8dGV4dGFyZWEgY29scz04MCByb3dzPTQwPgovL2Rp"+
5462 "Y3Zlcg1HU2hlbGGxcc0lNRVvxzZGljdGlvbmFyeVxzZm9yXHNXbm5Xbm5Xbm5jcy9vXHMyMODIwLTA4MzAK"+
5463 "R1NoZWWxsCUdTaGVsbArjgo/jgZ/jgZcJ56eBCndhdGFzaGkZ56eBCndhdGGFzaQnnp4EK44Gq"+
5464 "44G+44GICeWQjeWJjQpuYW1lhZQnlkI3liY0K44Gq44GL44GuCeS4remHjgpuYWthbm8J5JLit"+
5465 "6YeOCndhCeOCjwp0YQnjgZ8Kc2kJ44GXCnNoNoaQnjgZcKbm8J44GuCm5hCeOBqgptYQnjgb4K"+
5466 "ZQnjgYgKaGEEJ44GvCm5hCeOBqgprYQnjgYsKbm8J44GuCmRlCeOBpwpvzdQnjgZkKZVxzCWVj"+
5467 "aG8KZGljCWRpYRlplY2hvCWVjaG8KcmVwbGFjZQpyZXBsYXljGxheQpyZXBsYXQnVwZWF0CmR0CWRh"+
5468 "dGVcysnJVklbSVklLSVIOiVNOiVTJwp0aW9uCUXRpb24KJXQJJXQ5LY8gdG8g8gYmUgYW4gYW4g44GW0"+
5469 "aW9uCjwvdGV4dGFyZWE+Cg=="
5470 //</span>
5471
5472 var SumomoDic = //<span id="gsh-sumomo-dic">
5473 "data:text/dic;base64,"+
5474 "PG1ldGEgY2hhcnNldD0iVVRGGLTgiPgo8dGV4dGFyZWEgY29scz04MCByb3dzPTQwPgovL3Zl"+
5475 "cglHU2hlbGGxcc0lNRVvxzZGljdGlvbmFyeVxzZm9yXHNTdWlvbW9ccy8vvXHMyMDIwLTA4MzAK"+
5476 "c3UJ44GZCmCmlvCeOCggpubwnjga4KdQnjgYYKY2hpCeOBoQp0aQnjgaEKdWNoaXRRp"+
5477 "CeWGhQpzdWlvbW8J44GZ44KC44KCnNlbW9tb21vCeOBmeOCguOCguOBmeOCgueOCguOCeahdgwpt"+
5478 "b2lvbW8J5qGD44KCCiwsCeOAgQouLgnjgIIKPC90ZXh0YXJlYT4K"
5479 //</span>
5480
5481 var SijimiDic = //<span id="gsh-sijimi-dic">
5482 "data:text/dic;base64,"+
5483 "PG1ldGEgY2hhcnNldD0iVVRGGLTgiPgo8dGV4dGFyZWEgY29scz04MCByb3dzPTQwPgovL3Zl"+
5484 "cglHU2hlbGGxcc0lNRVvxzZGljdGlvbmFyeVxzZm9yXHNTaGlqaW1pLHMvMjAyMC0wODMw"+
5485 "CnNpCeOBlwpzaGGkJ44GXCmppCeOBmAptaQnjgb8KbmEJ44GqCmplCeOBmOOChQp4eXUJ44GF"+
5486 "CnUJ44GGCm5pCeOBqwprbwnjgZMKYnUJ44G2Cm5uCeOCkwpubwnjga4K2hpCeOBoQp0aQnj"+
5487 "gaEKa2EJ44GLCnJhCeOCiQosLAnjgIEKLi4J44CCCCnhuYW5hCeS4gwp4anVlCeWNgQp4bmkJ"+
5488 "5LqMCmtveAnlgIsKa29xceWAiwprb3gJ5YCLCm5hbmFgdXVuaXXgJNzIKbmFuYWp1dW5peeHgJ"+
5489 "77yX77ySCm5hbmFgdXVuaVgJ77yX77ySCuS4g+WNgeS6jHgJNzIKa29z2idW5uCeWAi+WIhgp0"+
5490 "aWthcFFxCeOBoeOBi+OCiQp0aWthcmJ5YqbCmnoaWthcmEJ5YqbCjwvdGV4dGFyZWE+Cg="
5491 //</span>
5492
5493 var JA_JKLDic = //<span id="gsh-ja-jkl-dic">
5494 "data:text/dic;base64,"+
5495 "Ly92ZXJsScU15SU1FamRpY2ptb3J2b3JVWpKQWpKKS0woMjAyMGowODE5KSheeLV4pL1NhdG94SVRT"+
5496 "CmtqamprBGtqa2tsa2psIOS4lueVjApqamtqamwJ44GCCmtqbAnjgYQYKa2tqbAnjgYYKamtq"+
5497 "amwJ44GICmtqa2trbAnjgYoKa2pra2pra2wJ44GLCmpramtrbAnjgYY0Ka2tramwJ44GPCmpramps"+
5498 "CeOBkQpqampqbAnjgZMKamtqa2psCeOBlQpqamtqa2wJ44GCCmpqamtqbAnjgZkKa2pqamts"+
5499 "CeOBmwpqampramprbAnjgZ0KamtsCeOBnwpra2prbAnjgaEKa2pqa2wJ44GkCmtqa2pqbAnjgaYK"+
```

```
5500  "a2tqa2tsCeOBqApramtsCeOBqgpqa2prbAnjgasKa2tra2wJ44GsCmpqa2psCeOBrQpra2pq"+
5501  "bAnjga4Kamtra2wJ44GvCmpqa2tqbAnjgbIKampra2wJ44GlCmtsCeOBuApqa2tsCeOBuwpq"+
5502  "a2tqbAnjgb4Ka2tqa2psCeOBvwpqbAnjgoAKamtra2psCeOCgQpqa2tqa2wJ44KCCmtqamwJ"+
5503  "44KECmpra2pqbAnjgoYKampsCeOCiApra2tsCeOCiQpqamtsCeOCigpqa2pqpqa2wJ44KLCmpq"+
5504  "amwJ44KMCmtqa2psCeOCjQpqa2psCeOCjwpramtramwJ44KQCmtqamtrbAnjgpEKa2pqamwJ"+
5505  "44KSCmtqa2prbAnjgpMKa2pqa2psCeODvApra2wJ44KbCmtramprbAnjgpwKa2pramtqbAnj"+
5506  "gIEK";
5507  //</span>
5508
5509  //</span>
5510  /*
5511  <details id="references"><summary>References</summary><div class="gsh-src">
5512  <p>
5513  <a href="https://golang.org">The Go Programming Language</a>
5514  <iframe src="https://golang.org" width="100%" height="300"></iframe>
5515
5516  <a href="https://developer.mozilla.org/ja/docs/Web">MDN web docs</a>
5517   <a href="https://developer.mozilla.org/ja/docs/Web/HTML/Element">HTML</a>
5518  CSS:
5519    <a href="https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Selectors">Selectors</a>
5520    <a href="https://developer.mozilla.org/en-US/docs/Web/CSS/background-repeat">repeat</a>
5521  HTTP
5522  JavaScript:
5523  ...
5524  </p>
5525  </div></details>
5526  */
5527  /*
5528  <details id="html-src" onclick="frame_open();"><summary>Raw Source</summary><div>
5529
5530  <!-- h2>The full of this HTML including the Go code is here.</h2 -->
5531  <details id="gsh-whole-view"><summary>Whole file</summary>
5532   <a name="whole-src-view"></a>
5533   <span id="src-frame"></span><!-- a window to show source code -->
5534  </details>
5535
5536  <details id="gsh-style-frame" onclick="fill_CSSView()"><summary>CSS part</summary>
5537   <a name="style-src-view"></a>
5538   <span id="gsh-style-view"></span>
5539  </details>
5540
5541  <details id="gsh-script-frame" onclick="fill_JavaScriptView()"><summary>JavaScript part</summary>
5542   <a name="script-src-view"></a>
5543   <span id="gsh-script-view"></span>
5544  </details>
5545
5546  <details id="gsh-data-frame" onclick="fill_DataView()"><summary>Builtin data part</summary>
5547   <a name="gsh-data-frame"></a>
5548   <span id="gsh-data-view"></span>
5549  </details>
5550
5551  </div></details>
5552  */
5553  /*
5554  <div id="gsh-footer" style=""></div><!-- ----------- END-OF-VISIBLE-PART ----------- -->
5555
5556
5557  <style id="gsh-style-def">
5558  //body {display:none;}
5559  .gsh-link{color:green;}
5560  #gsh {border-width:1;margin:0;padding:0;}
5561  #gsh {font-family:monospace,Courier New;color:#ddf;font-size:8px;}
5562  #gsh header{height:100px;}
5563  #xgsh header{height:100px;background-image:url(GShell-Logo00.png);}
5564  #gsh-menu{font-size:14pt;color:#f88;}
5565  #gsh-footer{height:100px;background-size:80px;background-repeat:no-repeat;}
5566  #gsh note{color:#000;font-size:10pt;}
5567  #gsh h2{color:#24a;font-family:Georgia;font-size:18pt;}
5568  #gsh h3{color:#24a;font-family:Georgia;font-size:16pt;}
5569  #gsh details{color:#888;background-color:#fff;font-family:monospace;}
5570  #gsh summary{font-size:16pt;color:#fff;background-color:#8af;height:30px;}
5571  #gsh pre{font-size:11pt;color:#223;background-color:#faffff;}
5572  #gsh a{color:#24a;}
5573  #gsh a[name]{color:#24a;font-size:16pt;}
5574  #gsh .gsh-src{white-space:pre;font-family:monospace,Courier New;font-size:11pt;}
5575  #gsh .gsh-src{background-color:#faffff;color:#223;}
5576  #gsh-src-src{spellcheck:false}
5577  #src-frame-textarea{white-space:pre;font-family:monospace,Courier New;font-size:11pt;}
5578  #src-frame-textarea{background-color:#faffff;color:#223;}
5579  .gsh-code {white-space:pre;font-family:monospace !import;}
5580  .gsh-code {color:#088;font-size:11pt; background-color:#eef;}
5581  .gsh-golang-data {display:none;}
5582  #gsh-WinId {color:#000;font-size:14pt;}
5583
5584  .gsh-document {font-size:11pt;background-color:#fff;font-family:Georgia;}
5585  .gsh-document {color:#000;background-color:#fff !import;}
5586  .gsh-document > h2{color:#000;background-color:#fff !import;}
5587  .gsh-document details{color:#000;background-color:#fff;font-family:Georgia;}
5588  .gsh-document p{max-width:550pt;color:#000;background-color:#fff;font-family:Georgia;}
5589  .gsh-document address{width:500pt;color:#000;background-color:#fff;font-family:Georgia;}
5590
5591  @media print {
5592    #gsh pre{font-size:11pt !import;}
5593  }
5594  </style>
5595
5596  <!--
5597  // Logo image should be drawn by JavaScript from a meta-font.
5598  // CSS seems not follow line-splitted URL
5599  -->
5600  <script id="gsh-data">
5601  //GshLogo="QR-ITS-more.jp.png"
5602  GshLogo="data:image/png;base64,\
5603  iVBORw0KGgoAAAANSUhEUgAAAQEAAAB/CAYAAADvs3f4AAAAAXNSR0IArs4c6QAAAHhlWElm\
5604  TU0AKgAAAAgABAEaAAUAAAABAAAAPgEbAAUAAAABAAAARgEoAAMAAAABAAIAAIdpAAQAAAAB\
5605  AAAATgAAAAAAAAABIAAAAAQAAAEgAAAABAAOgAQDAAAAAQABAACgAgEAAAAAAAAQGgAwAE\
5606  AAAAAQAAAH8AAAAAYx1BhgAAAAlwSFlzAAALEwAACxMBAJqcGAAAF3RJREFUeAHtnQuUFNWZ\
5607  x++t7ukZ3Z3Z3CiggO/jY6Osb8WgMzAvn7uG4+bISTR7Y7YnQXf7wGQ/PCkGj2aNwnw1D2mSM\
5608  4iuJx7jriYz50DOGmF2VqIBEiSggCoiMMMA+mu+vu//ZMD9U1dau2aUbv91OQrv91Gxrc6ReV\
5609  fnXvdx8tBA8tBA8SIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5610  IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5611  IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIIFDl4A8dLP2\
5612  2eXs9H9+ftSkSdHxsic2qqdE7YusS+1qaalKfnY5YsokMMMHwEPtdK4MQPz5UeElbLYSaYU15\
5613  npDiLKXEZZCl7FiRM5JSUaq9ScqcU6i+2kK3StuONy5reEGKJ1Qw7m7OoVKec8ToiQToghFS\
5614  jbOVHCstMRb3USXEJ8h6u7+xU4U4VVWFVAlAE/hcKGab9b2+xU4U4ZyPzpV 0N9DLmXpFK/6Op2P/Piyovf\
5615  N8mfM+/nJWNGnjw9KqQTo0VGSFt2p2Ril1gn3Oj3Vk0Vk7YsoWVMzEuVfPlRXYJdOak2 OTp0V65rOK\
5616  zrWocCOG6hvhgRaCj/dktj3d7XXAAE2GvNYCj/dkxj3d3Ul1gn3ij0Vk7YsoWVMzEuVfPlRYdf\
5617  L66as88pU/PN1pNlTLQJK7s73dPYxAr2iO+wPcC8QhbNcyhUU IlrryOTQvYYf5JfvqBL7jx\
5618  +cNHjBj5gRaCj/dktj3d7dXAXK6N4g8N0H4gKk6RS0zpYYzerqqS6RAoZ2ODQqf79P79FFN\
5619  L66as88pU/PN1pNlTLQJKJy73aPdXPSr20ur7iiwPcC8QhbNcyhUII1rryOTQvyyvdfqvBL7jx\
5620  +cNHjBj55gJRyDlJHy39o84D40H2Qtx8THaPeFuIOU+w1C+KnyhK5FGEv0OWGgAExB83eXMoLY\
5621  rikbd9gHEF52VgQ14h89FUA6kJyYFbbQbnzLJg4zFFesnDHCwvUoeiVQQb/5C9FY9D1U1Ue0H\
5622  +zGhUh9nSgQqrm0uWgurkI9RpjBD4Y6uQcQQD5TUOW63z3zMMHesy14V49isbdKyxbGHlCpFR\
5623  Uj6toACF7F9V9F58NBfDHT0MBaE74Ent+eWrrWr-Lz/QTw60AdB7QJUjps/OA7cOoBNBCeMUZ\
5624  ttCu/coG28fLpvKKElTPFV8jura RasEahbHvxaR1guoeBPyfUDo4+OfeBdyb8L4tz9Xe9SXFAMOc\
```

5625 bgGgov0g1zgGGw4jF392xnHhdc+Mwf3JTjfntZ2yC1YJBJXNUt5KIKyck1sxXRdld6BmcevN\
5626 aJovy/VBacMevqEP46/ZlnJjt9jx17VL53Zl5Mtvap1QGlNHw5pQDqXyNTQlZ2b8nGcMG2ZV\
5627 qOoFjSdYvV0AZzDfayidv6FJ35CS4jXZk9hir7e27zm6p3T8hLJpkYicJpVlHtK/DJFU4Jw1\
5628 lImhxM5IR9fzzgRKx4w/C+HQSPE+krbIyrN3qEPTNahsHaLDs2xh5Q5MCoPPVdEpgcqbm/8e\
5629 7/zdOaHptag/mlKJ77U0VG0xybTdX/Ex/PTfa/i7r7Ku+cSoiCxUwrohUxF16wEV9H+ccVgl\
5630 pd/CfU42AK2IUPlvTK1L/sJjyE5PVHqr728NzvfUzvvDODGy9GoopuuhmNLNfcTx48YHL2qH\
5631 f/8hpXVu/43rQg9xtq6YtcvlXDC3fmWDQn9nbf2le7wKE1bOK65icBu0Eqhd3IaW82dwKPUw\
5632 hrauc6ZcWdkcjUZK8EUXMae71zUqwCu2nbi6eVn1Ji9/P7eW+ioMAogF+NI3iJLSf8dn9ipA\
5633 WNW4rPy9jJxuPeDL/HXzNzgTsveslD2vsWHWI9mu5rvVvZX9foS4v/LfmqdEIpHDGlfM2uCW\
5634 gJIy2wOENPaZ3fEcivd+ZYNCNJYtrNyhyGAo8jRoJTAUmRiqOCJnRW5FpTN++frTwdh4SiUv\
5635 bVlWvbffLcRF04qazRD7176/rBjKylD5pBiZ5Wi4wQu7tikPBeCOpuW+Kj0sqP8GHNoAZuwL\
5636 iOzuywDhQ9zBr2xoDRqVQFi5QxxH6OwVjRKAAW46pvT+RxAJVLjW7vY9/+CeUBMkl68/rPQn\
5637 mCufKzaldFN/yI8gA5iwC3dkIKhsyvZuCYSVG/KHcwhFWDRKAMMcD8EKX+rHFl2A9bt2d172\
5638 2qNzOvzCDYmfEtNy7QogXDXWIKAIQ7cOQZchyADWnerqN5xVXttcJsdGp2OtwqmWJU7A+Eh7\
5639 yhYbUgm1IX7f7K1DwaRyUfN42FIuxNDdVEtamL6sYC9R26VtbZaW2px8Nfmehz3EM+mgsolk\
5640 d3/ZnBGElXPGUWzXg1YCq5eW5/zBGy54aWOgwWKfnWbqptcevWT4FUBvov32gew8DLzDTMaj\
5641 aupq7t/bMXX+yw/egJGKoTksy2d+gFBb9VoDvX5BlZTOR+Wfjyb0pP6U0XGOYNqR/quta3vB\
5642 Fgeua6qv2d7vn8dFdV3r1dBw34GSPg9i0DG9h5XWknh9kAaMmyJ6dklPzZmtD3cnu77vtw5C\
5643 h/YrG1p7Wxp/VvuRDuc+wsq54ymm+8zzKOgyRSPRa4IKoGzli8b6ytagcEPmb9v/m09cUATz\
5644 Jow6tVnPcMxHzj+sNNpHsCJyja6csrRsMyrGkiwF4I5UiouliLlRW7fmNLeX3z2+/GfWlLU2\
5645 Y572b6EAzkfYoPctJil5QlnJyLdrFrUZp1/3pmkuG/yN9gAoGyMTf7neVIvx/6CHUghlluh/\
5646 f9Uvo+gG7O3q7rzFL8xQ+zW+/8F6PW6fV7xSXhiNlayvWdz2X1ULm/4uL1mPwNoA5uGcdoL9\
5647 ZFa6cgoxzhTG6Q4lNR5Doj9xuvlcy+rFbcujVsnLkKv0CefphUbICLRMvl+9KP4vngHg6Fc2\
5648 NCqMSiCsnCkfxeD+mTflBwuxdmFbOZqT/194225Y3TCrzpQWhthG2zHraJO/yb0kkdhpanZq\
5649 GXwFF66/8Cb5AHcbzdpnhUjeG6YFow1gZeMmtqNCDekzTiXVuc3LK4yVTJepuq5tqSWFkXdA\
5650 ufu9MfWiG3sqnNtcX76+3xEXQWWzVeqSpvrZmC2afYSVy46l+O4KvyVgicCugG2rp0yPTveJ\
5651 o2Ulm2JWZEO+f6K0dFtNXfw2U9x7O/bqZct5z0Poi0+vdpyDJcdxrD34U9XCeHrloSktt3ug\
5652 AcwtkOO9FZFn+gWtWdS6ODcFoDrAxneOCfRXWUSoK93pBZXN7vAe+gwr506/2O4LXgngLbrC\
5653 76HgRdvetHz2WlMYVVqqm5zTTP5+7volRR/zJlOYlx+8ohOzEb+CV/0TU5ic3NGfjkSs30MZ\
5654 tFUtil+Yi4yfAcwkjzqpZyb6HlgJebwpgLYxoO9/j8k//WW3xS32gQPHrV5uMTjI1DFN2Op6\
5655 fz5ywF4HfmXD+/Buy4NVu73yEFbOK65icot+ZjP+8qf4JkYiTnGKTb/qST0zMKACq18jjPGL\
5656 A4PCxYNpMKOtjREv84HpyOsws/BsqyT2RGZ6rzl0gA9sBhEp46hsP2ratmOJeGrugBWDB2Pw\
5657 NYD1B4OSTMBmcmdS2E/GG2ZvrF7Uejsqyw/7A7guEH6Kyy19q3fpQQVgXtx4dz+Ueg+Lmy5v\
5658 bjjYtO+b5LSqpq5Nz6nwbFFhUdaYgemZy4ap1z5dlbByA3NQTC4F3RKYfOTkaUF9Xry0LwU8\
5659 sDMC/H29oV0GTNV1C+iZhTu27rgAebkb4+8H3P553qOOyu/WHj21ZWbd7z2XLuv4fA1gmQSV\
5660 2GML+6KmhorvaQWgne11yZ/glLX+IBNcn2FQ7F9Y5XQfN/qUa+Hr3UrAGg1MTLrG3bfPyEtp\
5661 m6d5oyCZcJmzX9nQ2jAqgbBymXSL9VzQSgBfxUBjHpbXbzM+vKueRBRiotE/Bw8ogf/LIZhY\
5662 /9Tcnsb68lt7DtgnQRE8lEvT2z9eWT5SjF71FSZoVlyfTLvqUTOb62etccbRO11HeS68SYeT\
5663 2OzUdegWmRTW7S7ng7dKrVi9rLztoMPBK73nA4YrdZfM+5DZsymDymaHnClokvPOVHG5FrQS\
5664 wCY6RwU9Dkx5MU9WQXMaX+ePguLw8/dvfg6UlLPvsPBpXspOniQwagElsm9gqNxctOEQlvj5\
5665 7tBBBjAdHkMPdY0/q/irWlbf44t5cNKQKwAq7DsuJzHl6Clz8bk+1u2u78FXYWFklQ4/qY2x\
5666 tYvjX8boyWN6zwc9/Ojwz7pUtv1Lp0NQ2UxLo8PKOdMu1uvooTDjLyxcrNWHEhjQWsyKrkPs\
5667 2JHl4LpJicQXoyp6nMs5fYsKeile0G95+WXcEj3m5mcmjNe5b+lyHZYELxGjRmDnY/HtMK0S\
5668 aPE7Md34PueUYz8DWDovSjzXVF/xsFe+Lpz/wjQQ9eiH94ZWqVS62+CUhV3lMtNjSHfXorHf\
5669 wKgZg9FwIrTCRJwjWh5+/ocSLzQlzG52BvItG+wOpqXRYeWcaRfrdbSgC5bD/PySxBHakPWO\
5670 qZx9y4L1OuABB4xk5we8qDsHO6++b0nwjzFXYaUViy6Ece0OlI7SAZkxOUgxtmZB9RcaVyxx\
5671 2CbMBjAdTcruWWyKriwy4myTH9zt3R93/8Xlj0ESWetyy7qFFj1odwkAmhFEA2KD6DlwNe6h\
5672 H52HuWwIaLQHQOUYZwr6yznTLs7rgu4OYBJq4JBWJCayRhTyeYx4X8/xCw+rus9L5yc50A+W\
5673 8v0w0N2ZxAw7VADPZcEDpXpdsLXoDKefrwEM+yj47aEAa7yxzMjXm+61FzUL46ch7cOd6Q/m\
5674 Wncf9BTvXbs6Z3hNxPIvmlkJhJUbTFkKRbaglQCWiwbuiiPtyKlhHwZaq8YKoeMcji9Iy9Ly\
5675 Pwk79U/55Bk75fSXMchwhj79Y35xY7qu8YspvTbqSG+55hdjjn6YS6ErfyqVOL2xoeLrbmWj\
5676 YwkqG5S2p1IOK5djzgs+2LB1B4Z6/gG+uosa6yuWOYljzcCuoG4llqxVQOYep1wu1xUL4pPR\
5677 zD3GL6wlVE4jA35xePk1NlSuBb/34RcwB6JXGgz6rflBBjBbJH7tlWbGDRVdb4bieXgpPbhN\
5678 NQT3iqMHz7ETHvuRxnv45r8FpfQWRnDiqVfV2qBlxEFl6+rqDLV82CTnVYBidBs2JfBpwMJP\
5679 aW3rXYbqm9qXMLnmChjCnvUN5fKMRc2LbzJBk8mU55cn4x/2rLdJQzNjtKkyuuO1pdqccfMz\
5680 gKGp/aHfXooVi+JTofimZuJyn8F7QHmhAMxdAaUeTX6c7F07sUUkgyq5Oz33vV/Z0C7b+scH\
5681 LtnpltH3YeW84ipGt4JWAnu7Pn5xwqjxB4IMabBc3Q8rfLzPCJfTc0SF0b8NaDzSFWqYfhBU\
5682 nm1djITHGhN3esSRt+42Mk5KWcTsxFMe35RJTvorP3rmn49VMOgfP8oiD19lX6IdvbXmkqjvb\
5683 NfydX9m8WimZlMLKZeSL/VzQSkDPzcdYcyte7lq/B4XKfKQaNeK3mL47r29fQL/gaT+/vrEO\
5684 gDTTX0U9UWbKUVMfh9MYuLZjVPzxxu0fPO0/pTedhOd/1XXxGZawfuXp6eGI1z+eme2X9lbo\
5685 0xuUll9F0bLaKGgQhafa5NVPhxjK7X0gLuOMRm+JAFefsnnaKzLRhZXLyBf5ediUwKc1/wD7\
5686 fD+JL72vEtDPEIqgWkZj6zFP/d5duzt+ZHihxfkLnhs7umT0llAjKkyVScenpJ1WAlAACzAE\
5687 dqV2Sx/S+nLN0dPelXVtD/SkUr+JL5/9VsbL75z+bYNS8Q2EuQN/Oa3x1/FJZS/VZ30EGcBg\
5688 ePdtCYCR0RCKr3q6vL0pOf7XfXvDAaVzcGjQECZX56CyYcmxZ/7CyuWar2IIN2xK4NOC075/\
5689 4yMTRk3XuwyfGJgmxt/xdbpt8uSRi7Fl1luoFJtQm3Ul7cKXfyqMVsfDvwpVq9RPAeh07FRv\
5690 hUL4693pwu1YyN+FX0C+Cy0VrIWXzylh/w3n7fiibreUtTsVURMitjpKWRYmPKkZmHDzFciM\
5691 dMflf6+eWl0/65lMmCDD2YFEl2dFycgj38aRAbQSPGX1sCGUcCaKrDOUyszauvgcZx6zAvTf\
5692 LLGqFlXPjFjyIthCkphR+cN+r76LoLJ1d3d45i+snDv9Yr4veCWg9+SrXtx6G/arezLXB4WX\
5693 tgzv7Wk4n+Z8f/FFzzUKIa3ky5ULmo9CE8N3HgLinI5IsRNy32hsXxoRnTBmBvWmiP9zT7o3\
5694 j0q8vnN35zecGfY1gCm1w2/fviCjoJXytieolL0xvRGhMyNZ1/IJtL6Ww3j5y8j+7i1dyU57\
5695 xLjDJmM+xOFQgtrucgEUTDVIpFcnovWAf2KAEvArG5T3tjBGQT+5rCIU+U1BzxPIPJumpRVP\
5696 4YEuz9wP9xlfvw/0ppuyxDp9uNPyih9l/XNXovNSd5dGG3C8wms31CzfrkCQUTCZSHj+wm8q\
5697 JV7XE3xM6WqjLSr6LVB668ToEXtHjJ/4Cdw24+uzFvsJrsT11RkFoOOALtznFZdf2SDl2QrQ\
5698 8YSV88pDsboVhRLQD6exvrEOj9y4g9DQPkC5Zmjjyz021LdV7yb3zfL8qmsDmoFARTVWFC3i\
5699 NlNQGwX1jEavqOMrZ78D2ZVefmHcdPfCU86nbFBB5rKFlfPMRHE6Fo0S0AtoVm/d8VV8km7D\
5700 C58YrseFuLvspLpbx79z64erdZNyuNLKileJdalUak7j0orr315x+YA9CbQBDF/ck7JkHDdB\
5701 E5sg69OKMH9pdRJd6v3vgEvYbdQcucSlVM9nO/QaPP3KZlve8zWCmJjk3OkX+30RKQE8KiwN\
5702 blxafhe29JqBL8of8GKam6n5P9mdGP5bmUikpmc22tR7BHSKjjP0kmCktCf/KAMlsOJXtejK\
5703 v7q+/OzmZbN/Z5IoHT3+NPgZn2eyx7uiZOJDM9xoyTcZBTOya+vndqW3URPijYxbmDOe1/au\
5704 zq4BrYqgslmphGdLIKxccmLwXskzBGwa94OstveB+sf714IiK3oiO5mXod+r9/I2VxB0P9Ec3\
5705 xp7XQYu8JGTqmcat0+NeY/99v3xbh+21bh03cnotljdfCZnzkeapSDN/vjDg4XP4Cnb8+W9p\
5706 9zzduKz2Q3fev05lytqtomo30pzk9Ec5sHOY+FXfV15Or6xr5HkDFMGAadKQ3yAO9DydFdjj\
5707 ppf5kjNq6qrnYi3DfyKI5h14oOKj1aZehBJ9NtWTfBAGvv1uIawS2xVTahfsB50dfrpseEaP\
5708 mRiFlXOm8Xm4xnP/fBy6aVg2fty5SkWno2mMPfSF3sgCf3o4UGGSj/wI548wVLfbVvab7Z0b\
5709 Xx/MrwGlf9ZrXPQMbMx5CiAfjiHIyXjhsR7BKkMfG8mLT+D3CdJF2qod1vNN3V3d60xW7hyf\
5710 koSVf0pEpkZFeqJWQtld70c6dnp1H7zi0z933hOLHWYJulREhZ7ptxeVe69XWH+3Jdasm6tO\
5711 iEWsY1G5j8Eaj2NR0adga7IeVOR2LBSCcVC8Z0u5Ue1JbspxVqHEcusjRKkYLW0VSSUinTmW\
5712 LaycfxHpSwIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIk\
5713 QAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIk\
5714 QAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIk\
5715 QAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIk\
5716 QAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIk\
5717 QAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAJ5Evh/ikTb\
5718 m38w0ncAAAAASUVORK5CYII=";
5719
5720 GshIcon="data:image/png;base64,\
5721 iVBORw0KGgoAAAANSUhEUgAAAKwAAAB/CAYAAAByMylZAAAAAXNSR0IArs4c6QAAAAHhlWElm\
5722 TU0AKgAAAAgABAEaAAUAAAABAAAAPgEbAAUAAAABAAAARgEoAAMAAAABAAIAAAIAdpAAQAAAAB\
5723 AAAATgAAAAAAAABIAAAAAQAAAEgAAAABAAOgAQADAAAAAQABAAAgAgAEAAAAAQAAAKygAwAE\
5724 AAAAAQAAAH8AAAAACt6tZwAAAAlwSFlzAAALEwAACxMBAJqcGAAADQRJREFUeAHtnQ9wwFNUd\
5725 x9/b21z+iYCKCiIK1amWlj/jH6BCkstFEfth1IGpRWdstQokQEunttrW2nFqOlYTlatinZ0\
5726 amdAqY6jIyOXi7kgg1arVv74b3BAQPkbVAjJ3e3r949AWcJpe93923+3723+3++3++3ff47\
5727 ffv+nxA8SIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESAE\
5728 SIAESIAESIAESIAESIAESIAESIAESIAESMCGgLRx84/Tylk/EYasHcANHxRK\
5729 XiEuf65tAHEwaD08ImP2wLTxTadyBmzrT+42zSRrd3peQvpxSMtrhgNYtyVzyH\
5730 ZUASIAESIAESOKII+LPR9dzsk5SELvxdBZTzlpQpQpkbIeFle+8Jan3AzkmkYyA/67BKXiek+pmQ\
5731 hsf7VweE6yPDZ+oM6JmAxwzznN6REVCCqC4SOXwihHiI1I8tXFFZcfcu0WdT8oMvYv1twOIjAI/\
5732 Nbqem3i3oooS8sa5044oUmz+EzcEAE/FWHxfnjo9FlD/YP80RNkNjRRRAVAWAlqEGGgE4c/gBGSE\
5733 rgK0AMb/d3uCJ1WJsIQFeFIAkNA8FeVYNSmWYWWJgR5F5x5N4tZBKw8lWvh5adBn3e/fwmuGtt\
5734 65pV5ccdz46QncHyziLl2ay 3ywTmXDa8cLcpLjlWOiolUl/s3dO5652zNqqBP\
5735 2D8HBZtDXp+28KXXicYYgq9jq9Fve6Eh5QVCVinOVEQkfL1Ka7gpKVKWbYUnFOodS8i4tKyWaGPT\
5736 e0Lc2e8+3+ra1plI62LEVYrnPs6YKfawSqSqSqWm0KfapW0pl162Bk9ILeLAHWWzi9\
5737 fawMFPi9QQt4cZ4yO9YoBVF9YCrE/2qnEo/ElFa4DDqDQHalosCUt+rpJzzsGLGGLGGJ56ZG5OQqVRtt\
5738 rHrDxjvvnShYMyysWPTg9GeZGEGGJeS2EWWmpj4skJ8/8VQAt/zSeLLuuz5aemlHziRVHZikhePdApAVH0\
5739 F6R5ZaZff851OgmVOrtSeXG/oTLB9s+r+h80iNhushhZ1Gf191F6l2/2cSNyTII2//wU0J8aEK\
5740 IX8/7zhymPt6cR3iboeH5qmPkp6jqGFCSCh/j8Jb++oZroEok6L/8ByhmLbSfC0+c/oH+iZ1GnfJ+0J\
5741 zFZ7hGNGqkbb3eRkVGnlmkGd2Yl4570lEgJWsXSjUhk5/aNZBhIebRd6i48iox9+NM/oiYtvLls4\
5742 4tZxwlBPIulbE6uGOgGbuT19JpJqKKn7ulu0rGnbrqGApyWITMdMoAVvO2duzN11Tn1yo4sapYoEocLy4\
5743 lBTb03ibUUBQ4yOU66pAN9sUnmfN9NNNSsCon5YN+f0BuNRofOkoRGtDhmrfv0TuYvzw29/5ewBv\
5744 CGjzpzLwdmWMGrf0IxDxucHa+vntq2fF1I10BQItOZYxfqvybAlQOJ 8sjXPp7yERv9fE\
5745 74raV8+EyC/6vWif5XAmnO5KqdX60Nlem+/QWOQAMM7mOeuBg2r+5BbbBav+wDPdlDdX8As337m\
5746 RbHvm2bRm7h95pJ1UL7n8xhJNevL1K+UuwWQK9cHcmG56VVGuMVJ7vTBbaUv/PTtZ9KTkxxuXxQe+E\
5747 sB0ngishHovWT/2FW5q6vw6VhwVWnvfp4QkxYqDwHyYQgHaIPS/eR.PdhIDX8FuW\
5748 lfhprLH6fTfbbP7VC6PfkKVFvwHsOZYsjzjzc1TK9rWJR1jecr0HtSIJhIu09BrBav6v9wYFZ0cKhI\
5749 5wZudRZKNx211qVU8p8pLmxuoBCaZpaVgvjdeIZO+SUopx3Xi6n3nH3ee0DUVaJ2Ko847e\

```
5750 GPqe4dUaP7R/74/WPD77y76+A3c574a/FyENPbyb9YB/cVZPn3oYfrtv3PCjGPJ0FAKqAet3\
5751 7T4wc6jEqpkHy0eaEKuDXFG7FWKKHh72Wx453a+vBKsbWtla7r0hpRYM9SyotQ1dvQtfd2/1\
5752 Tr/7DA8W5jK8WHVTLKfuMts4CszRV4I1A+Y8t/zD1L0now1VcTe7wfDHK3+TazxSTjKLi2K6\
5753 C8zV1gcGfhKsRNeUXnGQ9UBVoK390MFfZTUYZA8prA05RYnejKA0/huOtNw+cc5y9264nCLN\
5754 TyPHOlR+3pL9VWMIdCpG6p1LTstasnpJRcQ+BiH0q9kKGnrXmH4dRYnEzujfavZkBtINQKzX\
5755 eQd16tyHZZX6MCWt21h9JeY9+O/wj2AjrQ+hFTPfKYutZOqipvsrX7Oz6ZobW1yiJ0ePsfN3\
5756 csNYwSFsi3RXtKHi7ky7cCT+GEaorst0dzvHgMI/O9rVwtaHpulzsy0kf99UCZDBlzllHOqT\
5757 2yDWtdlsVHHx9fDrt1h1fOgMKMF/29W2qY7k7zDUuzlbutncUdLMKykR60OL45Oy2RSiuy8E\
5758 213vcxGbegYZDF3bH6gAT7f3yc0VArNdIoMx/886D1mVSBlTZPt5SDnaCMhXwpHmaf0Mmbfm\
5759 vhDsqJNGjXHr80QJu84lF/WeBp4PPAlZG1gtDlZu7VBWBRp9q/ubAB6EYVKYL/ulF8EXgsVc\
5760 V9eGEnbQ/DSrWOYsRxRiQB34EOx/ssYPD73WK9owbTpEezP++jfTSoqyoAzc6xR/ofj5QrDY\
5761 BnasW4Zhsv+2rDYr5qZQAvdp5We1t/GQSumZYW6HgmgfPJRo/y4aaU+7Gffy1+LS0KKWcC+3\
5762 AjzxxVwCLD+BYJ07RA6IHTuc8jclEpNNZZ5qZ4PS8xK09H9p55d2S3TmJNgu8zUPTN+OL/PC\
5763 dc2P4UFahmsfn47H6RP12VnwjzrZ5LufLwSLBs0YF72KosQJJyIzN2fL0OaGkG4U6b8+BxYQ\
5764 TkKVwenYqeupTgZ2ftH6qhg26/jB8aPKnoBo59jZZLh9L+O84E59USUQhki65Vwg6P3njyDW\
5765 85ziR500l+qAbRR6TsO+rzbMQxwv2xr0csSSmQI/fCFY7LPdZ2lJZr5KK+C5dELh6ixYITwL\
5766 Vl/nm4/cmBCW+nXmNWee48EZnelWaOf+EKyUroI1DOGpL3PawpZRGFp1nIhtCOXYQ5CLqPQW\
5767 RGj4fb1+LEuY3VncC8bZF4KVlszeZfVNVs6qjsQv++Y0t29BUzqWrjqWZDI1oBJWxzEl8vIx\
5768 KEFL9fdsBxp/2Xs6sgXKM3dfCLatfd8adBN1uOUNhlAfwg6Bw93sevqj1HMULPw/b14a6npg\
5769 pkSWlwr06fYMn+v3MlU6MwfbD3KwyWsTXwj2zUcuO4jSJ+6WUyjBTlLlpSv1okE327S/NJwX\
5770 MqJ+21W6VtW1Ti4TY/bUnDxmS7iu9baqa2OYX5DbUX1z9BRpGEvdrDHJ51k3m3z394VgdSYp\
5771 qZbnklkQbbVhBteHI61/0vu/ZgszaeFLR+tNOBCxY90Xa7q7BBtQ6tbuV/oYiPhu8xhzA4R7\
5772 o1MaOu3Q4pYZWHWwCt1aI7Ndi3bXo2P7v2p70cmmEPycew8L4Q6770Ev+htZPnED+mNPy/W2\
5773 9LRATHr5EJ/vQ5gfIlw7StTREMd4gAu5rQ3T6aRSqdmx7Z+/GB47ui290UWv9JX4AnJ7llIS\
5774 16Y+xi8sfm6YcrRQxul4K1ysH6Be9llOYHs79i/4cxbvgnH2jWB1jlXXxecYQuZU0g5WDluq\
5775 Y6xMFg2XRcb6wYrTJp5NO7ZuP3v9irmdNn4F5eSbKoHOtab6aStQOt7/beUgSubPmhrC27B1\
5776 0YQhS1OJvclkYo4fxKoZ+kqw+oajDVEsgVEr9PehP+SrXWnkMNLm6VpQnUiKxIzm+0PveQqf\
5777 h4F8J1j9WwOrt+64Stf50OWEzd2G5tCd/FZS/VXH3nagrQU1+4B2j8m8SsS/FmI9D3McBjwo\
5778 kRn3kXzuqzpsZkZU1cbOCRjmPWhQ1aBxM1gsduI3l5d3JlR9ywOVm9Np1kTiE/CQYIdtWZV2\
5779 8/KpqxnKkvc1bdveIDDt0Usc+RxmsDIpnxmIw78tYM6HZGdCt2fgZnJ+8xzuSRBvQ4zrhEw9\
5780 H926s8VJSOHFzRdII7AAPQwzkI7LsplurBj0QPxYbyb/8dmn2//ll/qqnago2AwqF/38+WEl\
5781 I4afr5Q5EXMARkAoI2CCP2xvJNV+lMZ78LkH3V27LWVt2n9w4/+6JqdkxJPLqd7b1TADkw1p\
5782 nIhS+QSI+HiEw5RPuVengV20d6Nf7K0t1oF1dj/ikUsatCEBEiABEiABEiABEiABEiABEiAB\
5783 EiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiAB\
5784 EiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiAB\
5785 EiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiAB\
5786 EiABEiABEiABEiABEiABEhD/B9wOq7SGUV++AAAAE1FTkSuQmCC";
5787
5788 ITSmoreQR="data:image/png;base64,\
5789 iVBORw0KGgoAAAANSUhEUgAAAG8AAABvAQMAAADYCwwjAAAABlBMVEX///9BaeFHqDaJAAAB\
5790 HklEQVQ4jdXTsa2EMAwGYCMX7sICkVgjXVaCBe7CArASXdaIlAWgS4HwM52EVS+mvSgS+ZBQ\
5791 8gcb4BdHyzwv8szMSaUBHNm+KAd4QC8LDpDn8ogT4UpPGci2jI8IGFx3x3a3e1LwPWaHknVyWecev\
5792 UEbDXaB0X2aNjueYDOzNklQassPCkjc4nW3E1SfwqYk6jU/vAkPhg0AlSFhve8Jt0dkwDMwr\
5793 yMGSSuPyWHHAr19k0tkV2sb3sdW2rUCqW88g4p4Rp1A9s1JPv9cTp1NRD4XAFKin8XaQCIwT6Lzq\
5794 ZO8dHw/4+U2Gzq1S8gbqVmkfr1N6YXK8QqlD00OmlGTMvzPERA8AL9vvbOifpSoL33fsVytrL\
5795 S9wiqDzznhUI38v5n783/gBuUs2eLg1c8gAAAABJRU5ErkJggg==";
```

```
5796
5797 </script>
5798
5799 <script id="gsh-script">
5800 //document.getElementById('gsh-faviconurl').href = GshIcon
5801 //document.getElementById('gsh-faviconurl').href = GshLogo
5802 document.getElementById('gsh-faviconurl').href = ITSmoreQR
5803
5804 // id of GShell HTML elemets
5805 var E_BANNER = "gsh-banner" // banner element in HTML
5806 var E_FOOTER = "gsh-footer" // footer element in HTML
5807 var E_GINDEX = "gsh-gindex" // index of Golang code of GShell
5808 var E_GOCODE = "gsh-gocode" // Golang code of GShell
5809 var E_TODO   = "gsh-todo"   // TODO of GShell
5810 var E_DICT   = "gsh-dict"   // Dictionaly of GShell
5811
5812 function bannerElem(){ return document.getElementById(E_BANNER); }
5813 function bannerStyleFunc(){ return bannerElem().style; }
5814 var bannerStyle = bannerStyleFunc()
5815 bannerStyle.backgroundImage = "url("+GshLogo+")";
5816
5817 function footerElem(){ return document.getElementById(E_FOOTER); }
5818 function footerStyle(){ return footerElem().sytle; }
5819 footerElem().style.backgroundImage="url("+ITSmoreQR+")";
5820 //footerStyle().backgroundImage = "url("+ITSmoreQR+")";
5821
5822 function html_fold(e){
5823     if( e.innerHTML == "Fold" ){
5824         e.innerHTML = "Unfold"
5825         document.getElementById('gsh-menu-exit').innerHTML=""
5826         document.getElementById('gsh-statement').open=false
5827         document.getElementById('html-src').open=false
5828         document.getElementById(E_GINDEX).open=false
5829         document.getElementById(E_GOCODE).open=false
5830         document.getElementById(E_TODO).open=false
5831         document.getElementById('references').open=false
5832     }else{
5833         e.innerHTML = "Fold"
5834         document.getElementById('gsh-statement').open=true
5835         document.getElementById(E_GINDEX).open=true
5836         document.getElementById(E_GOCODE).open=true
5837         document.getElementById(E_TODO).open=true
5838         document.getElementById('references').open=true
5839     }
5840 }
5841 function html_pure(e){
5842     if( e.innerHTML == "Pure" ){
5843         document.getElementById('gsh').style.display=true
5844         //document.style.display = false
5845         e.innerHTML = "Unpure"
5846     }else{
5847         document.getElementById('gsh').style.display=false
5848         //document.style.display = true
5849         e.innerHTML = "Pure"
5850     }
5851 }
5852
5853 var bannerIsStopping = false
5854 //NOTE: .com/JSREF/prop_style_backgroundposition.asp
5855 function shiftBG(){
5856     bannerIsStopping = !bannerIsStopping
5857     bannerStyle.backgroundPosition = "0 0";
5858 }
5859 // status should be inherited on Window Fork(), so use the status in DOM
5860 function html_stop(e,toggle){
5861     if( toggle ){
5862         if( e.innerHTML == "Stop" ){
5863             bannerIsStopping = true
5864             e.innerHTML = "Start"
5865         }else{
5866             bannerIsStopping = false
5867             e.innerHTML = "Stop"
5868         }
5869     }else{
5870         // update JavaScript variable from DOM status
5871         if( e.innerHTML == "Stop" ){ // shown if it's running
5872             bannerIsStopping = false
5873         }else{
5874             bannerIsStopping = true
```

```
5875              }
5876          }
5877  }
5878  html_stop(document.getElementById('gsh-menu-stop'),false) // onInit.
5879  //html_stop(bannerElem(),false) // onInit.
5880
5881  //https://www.w3schools.com/jsref/met_win_setinterval.asp
5882  function shiftBanner(){
5883      var now = new Date().getTime();
5884      //"console.log("now="+(now%10))
5885      if( !bannerIsStopping ){
5886          bannerStyle.backgroundPosition = ((now/10)%100000)+" 0";
5887      }
5888  }
5889  setInterval(shiftBanner,10); // onInit.
5890
5891  // <a href="https://developer.mozilla.org/ja/docs/Web/API/Window/open">window.open()</a>
5892  // from embedded html to standalone page
5893  var MyChildren = 0
5894  function html_fork(){
5895      MyChildren += 1
5896      WinId = document.getElementById('gsh-WinId').innerHTML + "." + MyChildren;
5897      newwin = window.open("",WinId,"");
5898      src = document.getElementById("gsh");
5899      newwin.document.write("/*<"+"html>\n");
5900      newwin.document.write("<"+"span id=\"gsh\">");
5901      newwin.document.write(src.innerHTML);
5902      newwin.document.write("<"+"/span><"+"/html>\n"); // gsh span
5903      newwin.document.getElementById('gsh-menu-exit').innerHTML = "Close";
5904      newwin.document.getElementById('gsh-WinId').innerHTML = WinId;
5905      newwin.document.close();
5906      newwin.focus();
5907  }
5908  function html_close(){
5909      window.close()
5910  }
5911  function win_jump(win){
5912      //win = window.top;
5913      win = window.openner; // https://developer.mozilla.org/ja/docs/Web/API/window.opener
5914      if( win == null ){
5915          console.log("jump to window.opener("+win+")(Error)\n")
5916      }else{
5917          console.log("jump to window.opener("+win+")\n")
5918          win.focus();
5919      }
5920  }
5921
5922  // 0.2.9 2020-0902 created chekcsum of HTML
5923  CRC32UNIX = 0x04C11DB7 // Unix cksum
5924  function byteCRC32add(bigcrc,octstr,octlen){
5925      var crc = new Int32Array(1)
5926      crc[0] = bigcrc
5927
5928      let oi = 0
5929      for( ; oi < octlen; oi++ ){
5930          var oct = new Int8Array(1)
5931          oct[0] = octstr[oi]
5932          for( bi = 0; bi < 8; bi++ ){
5933              //console.log("--CRC32 "+crc[0]+" "+oct[0].toString(16)+" ["+oi+"."+bi+"]\n")
5934              ovf1 = crc[0] < 0 ? 1 : 0
5935              ovf2 = oct[0] < 0 ? 1 : 0
5936              ovf = ovf1 ^ ovf2
5937              oct[0] <<= 1
5938              crc[0] <<= 1
5939              if( ovf ){ crc[0] ^= CRC32UNIX }
5940          }
5941      }
5942      //console.log("--CRC32 byteAdd return crc="+crc[0]+","+oi+"/"+octlen+"\n")
5943      return crc[0];
5944  }
5945  function strCRC32add(bigcrc,stri,strlen){
5946      var crc = new Uint32Array(1)
5947      crc[0] = bigcrc
5948      var code = new Uint8Array(strlen);
5949      for( i = 0; i < strlen; i++){
5950          code[i] = stri.charCodeAt(i) // not charAt() !!!!
5951          //console.log("=== "+code[i].toString(16)+" <<== "+stri[i]+"\n")
5952      }
5953      crc[0] = byteCRC32add(crc,code,strlen)
5954      //console.log("--CRC32 strAdd return crc="+crc[0]+"\n")
5955      return crc[0]
5956  }
5957  function byteCRC32end(bigcrc,len){
5958      var crc = new Uint32Array(1)
5959      crc[0] = bigcrc
5960      var slen = new Uint8Array(4)
5961      let li = 0
5962          for( ; li < 4; ){
5963              slen[li] = len
5964          li += 1
5965              len >>= 8
5966              if( len == 0 ){
5967                  break
5968              }
5969          }
5970      crc[0] = byteCRC32add(crc[0],slen,li)
5971      crc[0] ^= 0xFFFFFFFF
5972      return crc[0]
5973  }
5974  function strCRC32(stri,len){
5975      var crc = new Uint32Array(1)
5976      crc[0] = 0
5977      crc[0] = strCRC32add(0,stri,len)
5978      crc[0] = byteCRC32end(crc[0],len)
5979      //console.log("--CRC32 "+crc[0]+" "+len+"\n")
5980      return crc[0]
5981  }
5982  function html_cksum(){
5983      //alert("cksum="+strCRC32("",0))
5984      //alert("cksum="+strCRC32("0",1))
5985      //return
5986
5987      version = document.getElementById('gsh-version').innerHTML
5988      sfavico = document.getElementById('gsh-faviconurl').href;
5989      sbanner = document.getElementById('gsh-banner').style.backgroundImage;
5990      spositi = document.getElementById('gsh-banner').style.backgroundPosition;
5991      sfooter = document.getElementById('gsh-footer').style.backgroundImage;
5992      document.getElementById('gsh-faviconurl').href = "";
5993      document.getElementById('gsh-banner').style.backgroundImage = "";
5994      document.getElementById('gsh-banner').style.backgroundPosition = "";
5995      document.getElementById('gsh-footer').style.backgroundImage = ""
5996
5997      //html = document.getElementById("gsh").outerHTML;
5998      html = document.getElementById("gsh").innerHTML;
5999
```

```
6000            textarea = document.createElement("textarea")
6001            textarea.innerHTML = html
6002            // <a href="https://stackoverflow.com/questions/5796718/html-entity-decode">Thanks</a>
6003            text = textarea.value
6004            //textarea.destroy()
6005            text = ""
6006            + "/*<"+"html>\n"          // lost preamble text
6007            + "<"+"span id=\"gsh\">"    // lost preamble text
6008            + text
6009            + "<"+"/span><"+"/html>\n" // lost trail text
6010            ;
6011
6012        tlen = text.length
6013        console.log("length="+tlen+"\n"+text)
6014        alert("cksum : " + strCRC32(text,tlen) + " " + tlen + " " + version)
6015
6016        document.getElementById('gsh-faviconurl').href                = sfavico;
6017        document.getElementById('gsh-banner').style.backgroundImage    = sbanner;
6018        document.getElementById('gsh-banner').style.backgroundPosition = spositi;
6019        document.getElementById('gsh-footer').style.backgroundImage    = sfooter;
6020    }
6021
6022    // source code viewr
6023    function frame_close(){
6024        srcframe = document.getElementById("src-frame");
6025        srcframe.innterHTML = "";
6026        //srcframe.style.cols = 1;
6027        srcframe.style.rows = 1;
6028        srcframe.style.height = 0;
6029        srcframe.style.display = false;
6030        src = document.getElementById("src-frame-textarea");
6031        src.innerHML = ""
6032        //src.cols = 0
6033        src.rows = 0
6034        src.display = false
6035        //alert("--closed--")
6036    }
6037    //<!-- | <span onclick="html_view();">Source</span> -->
6038    //<!-- | <span onclick="frame_close();">SourceClose</span> -->
6039    //<!--| <span>Download</span> -->
6040    function frame_open(){
6041        document.getElementById('gsh-faviconurl').href = "";
6042        oldsrc = document.getElementById("GENSRC");
6043        if( oldsrc != null ){
6044            //alert("--I--(erasing old text)")
6045            oldsrc.innterHTML = "";
6046            return
6047        }else{
6048            //alert("--I--(no old text)")
6049        }
6050        banner = document.getElementById('gsh-banner').style.backgroundImage;
6051        footer = document.getElementById('gsh-footer').style.backgroundImage;
6052        document.getElementById('gsh-banner').style.backgroundImage = "";
6053        document.getElementById('gsh-banner').style.backgroundPosition = "";
6054        document.getElementById('gsh-footer').style.backgroundImage = "";
6055
6056        src = document.getElementById("gsh");
6057        srcframe = document.getElementById("src-frame");
6058        srcframe.innerHTML = ""
6059        + "<"+"cite id=\"GENSRC\">\n"
6060        + "<"+"style>\n"
6061        + "#GENSRC textarea{tab-size:4;}\n"
6062        + "#GENSRC textarea{-o-tab-size:4;}\n"
6063        + "#GENSRC textarea{-moz-tab-size:4;}\n"
6064        + "#GENSRC textarea{spellcheck:false;}\n"
6065        + "</"+"style>\n"
6066        + "<"+'textarea id="src-frame-textarea" cols=100 rows=20 class="gsh-code">'
6067        + "/*<"+"html>\n"               // lost preamble text
6068        + "<"+"span id=\"gsh\">"        // lost preamble text
6069        + src.innerHTML
6070        + "<"+"/span><"+"/html>\n"      // lost trail text
6071        + "</"+"textarea>\n"
6072        + "</"+"cite><!-- GENSRC -->\n";
6073
6074        //srcframe.style.cols = 80;
6075        //srcframe.style.rows = 80;
6076
6077        document.getElementById('gsh-banner').style.backgroundImage = banner;
6078        document.getElementById('gsh-footer').style.backgroundImage = footer;
6079    }
6080    function fill_CSSView(){
6081        part = document.getElementById('gsh-style-def')
6082        view = document.getElementById('gsh-style-view')
6083        view.innerHTML = ""
6084        + "<"+'textarea cols=100 rows=20 class="gsh-code">'
6085        + part.innerHTML
6086        + "<"+"/textarea>"
6087    }
6088    function fill_JavaScriptView(){
6089        jspart = document.getElementById('gsh-script')
6090        view = document.getElementById('gsh-script-view')
6091        view.innerHTML = ""
6092        + "<"+'textarea cols=100 rows=20 class="gsh-code">'
6093        + jspart.innerHTML
6094        + "<"+"/textarea>"
6095    }
6096    function fill_DataView(){
6097        part = document.getElementById('gsh-data')
6098        view = document.getElementById('gsh-data-view')
6099        view.innerHTML = ""
6100        + "<"+'textarea cols=100 rows=20 class="gsh-code">'
6101        + part.innerHTML
6102        + "<"+"/textarea>"
6103    }
6104    function jumpto_StyleView(){
6105        jsview = document.getElementById('html-src')
6106        jsview.open = true
6107        jsview = document.getElementById('gsh-style-frame')
6108        jsview.open = true
6109        fill_CSSView()
6110    }
6111    function jumpto_JavaScriptView(){
6112        jsview = document.getElementById('html-src')
6113        jsview.open = true
6114        jsview = document.getElementById('gsh-script-frame')
6115        jsview.open = true
6116        fill_JavaScriptView()
6117    }
6118    function jumpto_DataView(){
6119        jsview = document.getElementById('html-src')
6120        jsview.open = true
6121        jsview = document.getElementById('gsh-data-frame')
6122        jsview.open = true
6123        fill_DataView()
6124    }
```

```
6125  function jumpto_WholeView(){
6126      jsview = document.getElementById('html-src')
6127      jsview.open = true
6128      jsview = document.getElementById('gsh-whole-view')
6129      jsview.open = true
6130      frame_open()
6131  }
6132  function html_view(){
6133      html_stop();
6134
6135      banner = document.getElementById('gsh-banner').style.backgroundImage;
6136      footer = document.getElementById('gsh-footer').style.backgroundImage;
6137      document.getElementById('gsh-banner').style.backgroundImage = "";
6138      document.getElementById('gsh-banner').style.backgroundPosition = "";
6139      document.getElementById('gsh-footer').style.backgroundImage = "";
6140
6141      //srcwin = window.open("","CodeView2","");
6142      srcwin = window.open("","","");
6143      srcwin.document.write("<span id=\"gsh\">\n");
6144
6145      src = document.getElementById("gsh");
6146      srcwin.document.write("<"+"style>\n");
6147      srcwin.document.write("textarea{tab-size:4;}\n");
6148      srcwin.document.write("textarea{-o-tab-size:4;}\n");
6149      srcwin.document.write("textarea{-moz-tab-size:4;}\n");
6150      srcwin.document.write("</style>\n");
6151      srcwin.document.write("<h2>\n");
6152      srcwin.document.write("<"+"span onclick=\"window.close();\">Close</span> | \n");
6153      //srcwin.document.write("<"+"span onclick=\"html_stop();\">Run</span>\n");
6154      srcwin.document.write("</h2>\n");
6155      srcwin.document.write("<textarea id=\"gsh-src-src\" cols=100 rows=60>");
6156      srcwin.document.write("/*<"+"html>\n");
6157      srcwin.document.write("<"+"span id=\"gsh\">");
6158      srcwin.document.write(src.innerHTML);
6159      srcwin.document.write("<"+"/span><"+"/html>\n");
6160      srcwin.document.write("</"+"textarea>\n");
6161
6162      document.getElementById('gsh-banner').style.backgroundImage = banner;
6163      document.getElementById('gsh-footer').style.backgroundImage = footer
6164
6165      sty = document.getElementById("gsh-style-def");
6166      srcwin.document.write("<"+"style>\n");
6167      srcwin.document.write(sty.innerHTML);
6168      srcwin.document.write("<"+"/style>\n");
6169
6170      run = document.getElementById("gsh-script");
6171      srcwin.document.write("<"+"script>\n");
6172      srcwin.document.write(run.innerHTML);
6173      srcwin.document.write("<"+"/script>\n");
6174
6175      srcwin.document.write("<"+"/span><"+"/html>\n"); // gsh span
6176      srcwin.document.close();
6177      srcwin.focus();
6178  }
6179  </script>
6180  -->
6181  *///<br></span></html>
6182
```