

```

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="GshFaviconURL" href="#"><!-- place holder -->
6 <span id="GshVersion" style="display:none;">gsh--0.3.5--2020-09-08--SatoxITS</span>
7 <title>GShell - 0.3.5 by SatoxITS</title>
8 <header id="GshBanner" height="100px" onclick="shiftBG();" style="">
9 <div align="right"><note><a href="http://gshell.org">GShell</a> version 0.3.5 // 2020-09-08 // 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="GshMenu">
19 | <span id="gsh-menu-exit" onclick="html_close();"></span>
20 | <span id="gsh-menu-fork" onclick="html_fork();">Fork</span>
21 | <span id="GshMenuStop" onclick="html_stop(this,true);">Stop</span>
22 | <span id="GshMenuFold" onclick="html_fold(this);">Unfold</span>
23 | <span id="gsh-menu-cksum" onclick="html_digest();">Digest</span>
24 <!-- / <span id="gsh-menu-pure" onclick="html_pure(this);">Pure</span> -->
25 |</span>
26 */
27 /*
28 <details id="GshStatement" 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="GshFeatures" 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 <code>command mode</code></b></i> by <b>ESC</b> key,
62 then move around in the history by <b><code>j k l ? 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 redirecton
86   <a href="#history">history</a> // command history
87   <a href="#usage">usage</a> // resource 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
119 // <a name="import">Imported packages</a> // <a href="https://golang.org/pkg/">Packages</a>
120 import (
121   "fmt"      // <a href="https://golang.org/pkg/fmt/">fmt</a>
122   "strings"  // <a href="https://golang.org/pkg/strings/">strings</a>
123   "strconv"  // <a href="https://golang.org/pkg/strconv/">strconv</a>
124   "sort"     // <a href="https://golang.org/pkg/sort/">sort</a>

```

```

125     "time"      // <a href="https://golang.org/pkg/time/">time</a>
126     "bufio"     // <a href="https://golang.org/pkg/bufio/">bufio</a>
127     "io/ioutil" // <a href="https://golang.org/pkg/io/ioutil/">ioutil</a>
128     "os"        // <a href="https://golang.org/pkg/os/">os</a>
129     "syscall"   // <a href="https://golang.org/pkg/syscall/">syscall</a>
130     "plugin"    // <a href="https://golang.org/pkg/plugin/">plugin</a>
131     "net"       // <a href="https://golang.org/pkg/net/">net</a>
132     "net/http"  // <a href="https://golang.org/pkg/net/http/">http</a>
133     "html"      // <a href="https://golang.org/pkg/html/">html</a>
134     "path/filepath" // <a href="https://golang.org/pkg/path/filepath/">filepath</a>
135     "go/types"  // <a href="https://golang.org/pkg/go/types/">types</a>
136     "go/token"  // <a href="https://golang.org/pkg/go/token/">token</a>
137     "encoding/base64" // <a href="https://golang.org/pkg/encoding/base64/">base64</a>
138     "unicode/utf8" // <a href="https://golang.org/pkg/unicode/utf8/">utf8</a>
139     // "gshdata" // gshell's logo and source code
140     "hash/crc32" // <a href="https://golang.org/pkg/unicode/hash/crc32/">crc32</a>
141 )
142
143 // // 2020-0906 added,
144 // // <a href="https://golang.org/cmd/cgo/">CGo</a>
145 // #include <poll.h> // </poll.h> to be closed as HTML tag :-p
146 // typedef struct { struct pollfd fdv[8]; } pollFdV;
147 // int pollx(pollFdV *fdv, int nfds, int timeout);
148 // return poll(fdv->fdv, nfds,timeout);
149 //
150 import "C"
151
152 // // 2020-0906 added,
153 func CfpollIn1(fp*os.File, timeoutUs int)(ready uintptr){
154     var fdv = C.pollFdV{}
155     var nfds = 1
156     var timeout = timeoutUs/1000
157
158     fdv.fdv[0].fd = C.int(fp.Fd())
159     fdv.fdv[0].events = C.POLLIN
160     if( 0 < EventRecvFd ){
161         fdv.fdv[1].fd = C.int(EventRecvFd)
162         fdv.fdv[1].events = C.POLLIN
163         nfds += 1
164     }
165     r := C.pollx(&fdv,C.int(nfds),C.int(timeout))
166     if( r <= 0 ){
167         return 0
168     }
169     if (int(fdv.fdv[1].revents) & int(C.POLLIN)) != 0 {
170         //fprintf(stderr,"--De-- got Event\n");
171         return uintptr(EventFdOffset + fdv.fdv[1].fd)
172     }
173     if (int(fdv.fdv[0].revents) & int(C.POLLIN)) != 0 {
174         return uintptr(NormalFdOffset + fdv.fdv[0].fd)
175     }
176     return 0
177 }
178
179 const (
180     NAME = "gsh"
181     VERSION = "0.3.5"
182     DATE = "2020-09-08"
183     AUTHOR = "SatoxITS(^_)"
184 )
185 var {
186     GSH_HOME = ".gsh" // under home directory
187     GSH_PORT = 9999
188     MaxStreamSize = int64(128*1024*1024*1024) // 128GiB is too large?
189     PROMPT = "> "
190     LINESIZE = (8*1024)
191     PATHSEP = ":" // should be ";" in Windows
192     DIRSEP = "/" // canbe \ in Windows
193 )
194
195 // -xX logging control
196 // --A-- all
197 // --I-- info.
198 // --D-- debug
199 // --T-- time and resource usage
200 // --W-- warning
201 // --E-- error
202 // --F-- fatal error
203 // --Xn-- network
204
205 // <a name="struct">Structures</a>
206 type GCommandHistory struct {
207     StartAt    time.Time // command line execution started at
208     EndAt     time.Time // command line execution ended at
209     ResCode    int       // exit code of (external command)
210     CmdError   error    // error string
211     OutData   *os.File // output of the command
212     Foundfile []string // output - result of ufind
213     Rusageev  [2]syscall.Rusage // Resource consumption, CPU time or so
214     CmdId     int       // maybe with identified with arguments or impact
215     // redirection commands should not be the CmdId
216     WorkDir   string    // working directory at start
217     WorkDirX  int       // index in ChdirHistory
218     Cmdline   string    // command line
219 }
220 type GCkdirHistory struct {
221     Dir      string
222     Movedat  time.Time
223     CmdIndex int
224 }
225 type CmdMode struct {
226     BackGround bool
227 }
228 type Event struct {
229     when    time.Time
230     event   int
231     evarg   int64
232     CmdIndex int
233 }
234 var CmdIndex int
235 var Events []Event
236 type PluginInfo struct {
237     Spec      *plugin.Plugin
238     Addr      plugin.Symbol
239     Name      string // maybe relative
240     Path      string // this is in Plugin but hidden
241 }
242 type GServer struct {
243     host      string
244     port      string
245 }
246
247 // <a href="https://tools.ietf.org/html/rfc3230">Digest</a>
248 const ( // SumType
249     SUM_ITEMS = 0x000001 // items count

```

```

250     SUM_SIZE          = 0x000002 // data length (simply added)
251     SUM_SIZEHASH      = 0x000004 // data length (hashed sequence)
252     SUM_DATEHASH      = 0x000008 // date of data (hashed sequence)
253     // also envelope attributes like time stamp can be a part of digest
254     // hashed value of sizes or mod-date of files will be useful to detect changes
255
256     SUM_WORDS          = 0x000010 // word count is a kind of digest
257     SUM_LINES          = 0x000020 // line count is a kind of digest
258     SUM_SUM64          = 0x000040 // simple add of bytes, useful for human too
259
260     SUM_SUM32_BITS      = 0x000100 // the number of true bits
261     SUM_SUM32_2BYTE    = 0x000200 // 16bits words
262     SUM_SUM32_4BYTE    = 0x000400 // 32bits words
263     SUM_SUM32_8BYTE    = 0x000800 // 64bits words
264
265     SUM_SUM16_BSD       = 0x001000 // UNIXsum -sum -bsd
266     SUM_SUM16_SYSV      = 0x002000 // UNIXsum -sum -sysv
267     SUM_UNIXFILE        = 0x004000
268     SUM_CRCIEEE         = 0x008000
269 }
270 type CheckSum struct {
271     Files           int64 // the number of files (or data)
272     Size            int64 // content size
273     Words           int64 // word count
274     Lines           int64 // line count
275     SumType         int
276     Sum64          uint64
277     Crc32Table     crc32.Table
278     Crc32Val       uint32
279     Sum16          int
280     Ctime          time.Time
281     Atime          time.Time
282     Mtime          time.Time
283     Start           time.Time
284     Done            time.Time
285     RusageAtStart [2]syscall.Rusage
286     RusageAtEnd   [2]syscall.Rusage
287 }
288 type ValueStack [][]string
289 type GshContext struct {
290     Startdir        string // the current directory at the start
291     Getline         string // gsh-getline command as a input line editor
292     ChdirHistory   []GChdirHistory // the 1st entry is wd at the start
293     gshPA          syscall.ProcAttr
294     CommandHistory []GCommandHistory
295     CmdCurrent     GCommandHistory
296     BackGround     bool
297     BackGroundJobs []int
298     LastRusage     syscall.Rusage
299     GshHomeDir     string
300     TerminalId     int
301     CmdDtrace      bool // should be [map]
302     CmdDtime       bool // should be [map]
303     PluginFuncs   []PluginInfo
304     iValues         []string
305     iDelimiter      string // field separator of print out
306     iFormat         string // default print format (of integer)
307     iValStack      ValueStack
308     LastServer     GServer
309     RSERV          string // [gsh://]host[:port]
310     RWD            string // remote (target, there) working directory
311     lastCheckSum   CheckSum
312 }
313
314 func nsleep(ns time.Duration){
315     time.Sleep(ns)
316 }
317 func usleep(ns time.Duration){
318     nsleep(ns*1000)
319 }
320 func msleep(ns time.Duration){
321     nsleep(ns*1000000)
322 }
323 func sleep(ns time.Duration){
324     nsleep(ns*1000000000)
325 }
326
327 func strBegins(str, pat string)(bool){
328     if len(pat) <= len(str){
329         yes := str[0:len(pat)] == pat
330         //fmt.Printf("--D-- strBegins(%v,%v)=%v\n",str,pat,yes)
331         return yes
332     }
333     //fmt.Printf("--D-- strBegins(%v,%v)=%v\n",str,pat,false)
334     return false
335 }
336 func isin(what string, list []string) bool {
337     for _, v := range list {
338         if v == what {
339             return true
340         }
341     }
342     return false
343 }
344 func isinX(what string,list[]string)(int{
345     for i,v := range list {
346         if v == what {
347             return i
348         }
349     }
350     return -1
351 }
352
353 func env(opts []string) {
354     env := os.Environ()
355     if isin("-s", opts){
356         sort.Slice(env, func(i,j int) bool {
357             return env[i] < env[j]
358         })
359     }
360     for _, v := range env {
361         fmt.Printf("%v\n",v)
362     }
363 }
364
365 // - rewriting should be context dependent
366 // - should postpone until the real point of evaluation
367 // - should rewrite only known notation of symbol
368 func scanInt(str string)(val int,leng int){
369     leng = -1
370     for i,ch := range str {
371         if '0' <= ch && ch <= '9' {
372             leng = i+1
373         }else{
374             break

```

```

375     }
376 }
377 if 0 < leng {
378     ival,_ := strconv.Atoi(str[0:leng])
379     return ival,leng
380 }else{
381     return 0,0
382 }
383 }
384 func substHistory(gshCtx *GshContext,str string,i int,rstr string)(leng int,rst string){
385 if len(str[i+1:]) == 0 {
386     return 0,rstr
387 }
388 hi := 0
389 histlen := len(gshCtx.CommandHistory)
390 if str[i+1] == '!' {
391     hi = histlen - 1
392     leng = 1
393 }else{
394     hi,leng = scanInt(str[i+1:])
395     if leng == 0 {
396         return 0,rstr
397     }
398     if hi < 0 {
399         hi = histlen + hi
400     }
401 }
402 if 0 <= hi && hi < histlen {
403     var ext byte
404     if 1 < len(str[i+leng:]) {
405         ext = str[i+leng:]|1]
406     }
407     //fmt.Printf("--D-- %v(%c)\n",str[i+leng:],str[i+leng])
408     if ext == 'f' {
409         leng += 1
410         xlist := []string{}
411         list := gshCtx.CommandHistory[hi].FoundFile
412         for _,v := range list {
413             //list[i] = escapeWhiteSP(v)
414             xlist = append(xlist,escapeWhiteSP(v))
415         }
416         //rstr += strings.Join(list, " ")
417         rstr += strings.Join(xlist," ")
418     }else
419     if ext == 'e' || ext == 'd' {
420         // !N@.. workdir at the start of the command
421         leng += 1
422         rstr += gshCtx.CommandHistory[hi].WorkDir
423     }else{
424         rstr += gshCtx.CommandHistory[hi].CmdLine
425     }
426 }else{
427     leng = 0
428 }
429 return leng,rstr
430 }
431 func escapeWhiteSP(str string)(string){
432 if len(str) == 0 {
433     return "\z" // empty, to be ignored
434 }
435 rstr := ""
436 for _,ch := range str {
437     switch ch {
438     case '\\': rstr += "\\\\"
439     case ':': rstr += "\\s"
440     case '\t': rstr += "\\t"
441     case '\r': rstr += "\\r"
442     case '\n': rstr += "\\n"
443     default: rstr += string(ch)
444     }
445 }
446 return rstr
447 }
448 func unescapeWhiteSP(str string)(string){ // strip original escapes
449 rstr := ""
450 for i := 0; i < len(str); i++ {
451     ch := str[i]
452     if ch == '\\' {
453         if i+1 < len(str) {
454             switch str[i+1] {
455             case 'z':
456                 continue;
457             }
458         }
459     }
460     rstr += string(ch)
461 }
462 return rstr
463 }
464 func unescapeWhiteSPV(strv []string)([]string){ // strip original escapes
465 ustrv := []string{}
466 for _,v := range strv {
467     ustrv = append(ustrv,unescapeWhiteSP(v))
468 }
469 return ustrv
470 }
471 // <a name="comexpansion">str-expansion</a>
472 // - this should be a macro processor
473 func strsubst(gshCtx *GshContext,str string,histonly bool) string {
474     rbuf := []byte{}
475     if false {
476         //@U Unicode should be cared as a character
477         return str
478     }
479     //rstr := ""
480     inEsc := 0 // escape characer mode
481     for i := 0; i < len(str); i++ {
482         //fmt.Printf("--D--Subst %v:%v\n",i,str[i:])
483         ch := str[i]
484         if inEsc == 0 {
485             if ch == '\'' {
486                 //leng,xrstr := substHistory(gshCtx,str,i,rstr)
487                 leng,rs := substHistory(gshCtx,str,i,"")
488                 if 0 < leng {
489                     //_rs := substHistory(gshCtx,str,i,"")
490                     rbuf = append(rbuf,[]byte(rs)...)
491                     i += leng
492                     //rstr = xrstr
493                     continue
494                 }
495             }
496             switch ch {
497             case '\\': inEsc = '\\'; continue
498             //case '%': inEsc = '%'; continue
499         }
500     }
501 }
```

```

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

```

```

625     if count <= 0 {
626         break
627     }
628     if err != nil {
629         break
630     }
631     os.Stdout.Write(buff[0:count])
632 }
633 // lnsp [N] [-crlf][[-C \\]]
634 func (gshctx *GshContext)SplitLine(argv[]string){
635     strRep := isin("-str",argv) // "..."+
636     reader := bufio.NewReaderSize(os.Stdin,64*1024)
637     ni := 0
638     ni := 0
639    toi := 0
640     for ni = 0; ; ni++ {
641         line, err := reader.ReadString('\n')
642         if len(line) <= 0 {
643             if err != nil {
644                 fmt.Fprintf(os.Stderr,"--I-- lnsp %d to %d (%v)\n",ni,toi,err)
645                 break
646             }
647         }
648         off := 0
649         ilen := len(line)
650         remlen := len(line)
651         if strRep { os.Stdout.Write([]byte("\n")) }
652         for oi := 0; 0 < remlen; oi++ {
653             olen := remlen
654             addnl := false
655             if 72 < olen {
656                 olen = 72
657                 addnl = true
658             }
659             fmt.Fprintf(os.Stderr,"--D-- write %d [%d.%d] %d %d/%d/%d\n",
660                         toi,ni,oi,off,olen,remlen,ilen)
661            toi += 1
662             os.Stdout.Write([]byte(line[:olen]))
663             if addnl {
664                 if strRep {
665                     os.Stdout.Write([]byte("\r\n"))
666                 }else{
667                     //os.Stdout.Write([]byte("\r\n"))
668                     os.Stdout.Write([]byte("\n"))
669                     os.Stdout.Write([]byte("\n"))
670                 }
671             line = line[olen:]
672             off += olen
673             remlen -= olen
674         }
675         if strRep { os.Stdout.Write([]byte("\n")) }
676     }
677     fmt.Fprintf(os.Stderr,"--I-- lnsp %d to %d\n",ni,toi)
678 }
679 }
680 // CRC32 <a href="http://golang.jp/pkg/hash-crc32">crc32</a>
681 // 1 0000 0100 1100 0001 1011 1011 0111
682 var CRC32UNIX uint32 = uint32(0x04C11DB7) // Unix cksum
683 var CRC32IEEE uint32 = uint32(0xEDB88320)
684 func byteCRC32add(crc uint32,str[]byte,len uint64)(uint32){
685     var oi uint64
686     for oi = 0; oi < len; oi++ {
687         var oct = str[oi]
688         for bi := 0; bi < 8; bi++ {
689             //fprintf(stderr,"--CRC32 %d %X (%d.%d)\n",crc,oct,oi,bi)
690             ovf1 := (crc & 0x80000000) != 0
691             ovf2 := (oct & 0x80) != 0
692             ovf := (ovf1 && !ovf2) || (!ovf1 && ovf2)
693             oct <<= 1
694             crc <<= 1
695             if ovf { crc ^= CRC32UNIX }
696         }
697     }
698     //fprintf(stderr,"--CRC32 return %d %d\n",crc,len)
699     return crc;
700 }
701 }
702 func byteCRC32end(crc uint32, len uint64)(uint32){
703     var slen = make([]byte,4)
704     var li = 0
705     for li = 0; li < 4; {
706         slen[li] = byte(len)
707         li += 1
708         len >>= 8
709         if( len == 0 ){
710             break
711         }
712     }
713     crc = byteCRC32add(crc,slen,uint64(li))
714     crc ^= 0xFFFFFFFF
715     return crc
716 }
717 func strCRC32(str string,len uint64)(crc uint32){
718     crc = byteCRC32add(0,[]byte(str),len)
719     crc = byteCRC32end(crc,len)
720     //fprintf(stderr,"--CRC32 %d %d\n",crc,len)
721     return crc
722 }
723 func CRC32Finish(crc uint32, table *crc32.Table, len uint64)(uint32){
724     var slen = make([]byte,4)
725     var li = 0
726     for li = 0; li < 4; {
727         slen[li] = byte(len & 0xFF)
728         li += 1
729         len >>= 8
730         if( len == 0 ){
731             break
732         }
733     }
734     crc = crc32.Update(crc,table,slen)
735     crc ^= 0xFFFFFFFF
736     return crc
737 }
738 func (gsh*GshContext)xCksum(path string,argv[]string, sum*CheckSum)(int64){
739     if isin("-type/f",argv) && !IsRegFile(path){
740         return 0
741     }
742     if isin("-type/d",argv) && IsRegFile(path){
743         return 0
744     }
745     file, err := os.OpenFile(path,os.O_RDONLY,0)
746     if err != nil {
747         fmt.Printf("--E-- cksum %v (%v)\n",path,err)
748         return -1
749     }

```

```

750 }
751 defer file.Close()
752 if gsh.CmdTrace { fmt.Printf("--I--- cksum %v %v\n",path,argv) }
753
754 bi := 0
755 var buff = make([]byte,32*1024)
756 var total int64 = 0
757 var initTime = time.Time{}
758 if sum.Start == initTime {
759     sum.Start = time.Now()
760 }
761 for bi = 0; ; bi++ {
762     count,err := file.Read(buff)
763     if count <= 0 || err != nil {
764         break
765     }
766     if (sum.SumType & SUM_SUM64) != 0 {
767         s := sum.Sum64
768         for _,c := range buff[0:count] {
769             s += uint64(c)
770         }
771         sum.Sum64 = s
772     }
773     if (sum.SumType & SUM_UNIXFILE) != 0 {
774         sum.Crc32Val = byteCRC32add(sum.Crc32Val,buff,uint64(count))
775     }
776     if (sum.SumType & SUM_CRCIEEE) != 0 {
777         sum.Crc32Val = crc32.Update(sum.Crc32Val,sum.Crc32Table,buff[0:count])
778     }
779 // <a href="https://en.wikipedia.org/wiki/BSD_checksum">BSD checksum</a>
780 if (sum.SumType & SUM_SUM16_BSD) != 0 {
781     s := sum.Sum16
782     for _,c := range buff[0:count] {
783         s = (s >> 1) + ((s & 1) << 15)
784         s += int(c)
785         s &= 0xFFFF
786         //fmt.Printf("BSDsum: %d[%d] %d\n",sum.Size+int64(i),i,s)
787     }
788     sum.Sum16 = s
789 }
790 if (sum.SumType & SUM_SUM16_SYSV) != 0 {
791     for bj := 0; bj < count; bj++ {
792         sum.Sum16 += int(buff[bj])
793     }
794 }
795 total += int64(count)
796 }
797 sum.Done = time.Now()
798 sum.Files += 1
799 sum.Size += total
800 if !isin("-s",argv) {
801     fmt.Printf("%v ",total)
802 }
803 return 0
804 }
805
806 // <a name="grep">grep</a>
807 // "lines", "lin" or "lnp" for "(text) line processor" or "scanner"
808 // a*,lab,c, ... sequential combination of patterns
809 // what "LINE" is should be definable
810 // generic line-by-line processing
811 // grep [-v]
812 // cat -n -v
813 // uniq [-c]
814 // tail -f
815 // sed s/x/y/ or awk
816 // grep with line count like wc
817 // rewrite contents if specified
818 func (gsh*GshContext)xGrep(path string,rexpv[]string)(int){
819     file, err := os.OpenFile(path,os.O_RDONLY,0)
820     if err != nil {
821         fmt.Printf("--E-- grep %v (%v)\n",path,err)
822         return -1
823     }
824     defer file.Close()
825     if gsh.CmdTrace { fmt.Printf("--I--- grep %v %v\n",path,rexpv) }
826     //reader := bufio.NewReaderSize(file,LINESIZE)
827     reader := bufio.NewReaderSize(file,80)
828     li := 0
829     found := 0
830     for li = 0; ; li++ {
831         line, err := reader.ReadString('\n')
832         if len(line) <= 0 {
833             break
834         }
835         if 150 < len(line) {
836             // maybe binary
837             break;
838         }
839         if err != nil {
840             break
841         }
842         if 0 <= strings.Index(string(line),rexpv[0]) {
843             found += 1
844             fmt.Printf("%s:%d: %s",path,li,line)
845         }
846     }
847     //fmt.Printf("total %d lines %s\n",li,path)
848     //if( 0 < found){ fmt.Printf("(found %d lines %s)\n",found,path); }
849     return found
850 }
851
852 // <a name="finder">Finder</a>
853 // finding files with it name and contents
854 // file names are ORed
855 // show the content with %x fmt list
856 // ls -R
857 // tar command by adding output
858 type fileSum struct {
859     Err int64 // access error or so
860     Size int64 // content size
861     DupSize int64 // content size from hard links
862     Blocks int64 // number of blocks (of 512 bytes)
863     DupBlocks int64 // Blocks pointed from hard links
864     HLinks int64 // hard links
865     Words int64
866     Lines int64
867     Files int64
868     Dirs int64 // the num. of directories
869     Symlink int64
870     Flats int64 // the num. of flat files
871     MaxDepth int64
872     MaxNameLen int64 // max. name length
873     nextRepo time.Time
874 }

```

```

875 func showUsage(dir string,fusage *fileSum){
876     bsum := float64((fusage.Blocks-fusage.DupBlocks)/2)*1024)/1000000.0
877     //bsumdup := float64((fusage.Blocks/2)*1024)/1000000.0
878
879     fmt.Printf("v: %v files (%vd %vs %vh) %.6f MB (%.2f MBK)\n",
880             dir,
881             fusage.Files,
882             fusage.Dirs,
883             fusage.Symlink,
884             fusage.HLinks,
885             float64(fusage.Size)/1000000.0,bsum);
886 }
887 const (
888     S_IFMT    = 0170000
889     S_IFCHR   = 0020000
890     S_IFDIR   = 0040000
891     S_IFREG   = 0100000
892     S_IFLNK   = 0120000
893     S_IFSOCK  = 0140000
894 )
895 func cumFinfo(fsum *fileSum, path string, staterr error, fstat syscall.Stat_t, argv[]string,verb bool)(*fileSum){
896     now := time.Now()
897     if time.Second <= now.Sub(fsum.nextRepo) {
898         if !fsum.nextRepo.IsZero(){
899             tstamp := now.Format(time.Stamp)
900             showFusage(tstamp,fsum)
901         }
902         fsum.nextRepo = now.Add(time.Second)
903     }
904     if staterr != nil {
905         fsum.Err += 1
906         return fsum
907     }
908     fsum.Files += 1
909     if l < fstat.Nlink {
910         // must count only once...
911         // at least ignore ones in the same directory
912         //if finfo.Mode().IsRegular() {
913         if (fstat.Mode & S_IFMT) == S_IFREG {
914             fsum.HLinks += 1
915             fsum.DupBlocks += int64(fstat.Blocks)
916             //fmt.Printf("==>Dup HardLink %v %s\n",fstat.Nlink,path)
917         }
918     }
919     //fsum.Size += finfo.Size()
920     fsum.Size += fstat.Size
921     fsum.Blocks += int64(fstat.Blocks)
922     //if verb { fmt.Printf("(%dBlk) %s",fstat.Blocks/2,path) }
923     if isin("-ls",argv){
924         //if verb { fmt.Printf("%d %d ",fstat.Blksize,fstat.Blocks) }
925         fmt.Printf("%dt",fstat.Blocks/2)
926     }
927     //if finfo.IsDir()
928     if (fstat.Mode & S_IFDIR) == S_IFDIR {
929         fsum.Dirs += 1
930     }
931     //if (finfo.Mode() & os.ModeSymlink) != 0
932     if (fstat.Mode & S_IFMT) == S_IFLNK {
933         //if verb { fmt.Printf("symlink(%v,%s)\n",fstat.Mode,finfo.Name()) }
934         // fmt.Printf("symlink(%o,%s)\n",fstat.Mode,finfo.Name())
935         fsum.Symlink += 1
936     }
937     return fsum
938 }
939 func (gsh*GshContext)xxFindEntv(depth int,total *fileSum,dir string, dstat syscall.Stat_t, ei int, entv []string,npatv[]string,argv[]string)(*fileSum){
940     nols := isin("-grep",argv)
941     // sort entv
942     /*
943     if isin("-t",argv){
944         sort.Slice(filev, func(i,j int) bool {
945             return 0 < filev[i].ModTime().Sub(filev[j].ModTime())
946         })
947     */
948     /*
949     if isin("-u",argv){
950         sort.Slice(filev, func(i,j int) bool {
951             return 0 < filev[i].AccTime().Sub(filev[j].AccTime())
952         })
953     }
954     if isin("-U",argv){
955         sort.Slice(filev, func(i,j int) bool {
956             return 0 < filev[i].CreatTime().Sub(filev[j].CreatTime())
957         })
958     */
959     /*
960     if isin("-S",argv){
961         sort.Slice(filev, func(i,j int) bool {
962             return filev[j].Size() < filev[i].Size()
963         })
964     */
965     for _,filename := range entv {
966         for _,npat := range npatv {
967             match := true
968             if npat == "*" {
969                 match = true
970             }else{
971                 match, _ = filepath.Match(npatt,filename)
972             }
973             path := dir + DIRSEP + filename
974             if !match {
975                 continue
976             }
977             var fstat syscall.Stat_t
978             staterr := syscall.Lstat(path,&fstat)
979             if staterr != nil {
980                 if !isin("-w",argv){fmt.Printf("ufind: %v\n",staterr) }
981                 continue;
982             }
983             if isin("-du",argv) && (fstat.Mode & S_IFMT) == S_IFDIR {
984                 // should not show size of directory in "-du" mode ...
985             }else
986             if !nols && !isin("-h",argv) && (!isin("-du",argv) || isin("-a",argv)) {
987                 if isin("-du",argv) {
988                     fmt.Printf("%dt",fstat.Blocks/2)
989                 }
990                 showFileInfo(path,argv)
991             }
992             if true { // && isin("-du",argv)
993                 total = cumFinfo(total,path,staterr,fstat,argv,false)
994             }
995             /*
996             if isin("-wc",argv) {
997             */
998         }
999     }

```

```

1000 }
1001 */
1002 if gsh.lastCheckSum.SumType != 0 {
1003     gsh.xCksum(path,argv,&gsh.lastCheckSum);
1004 }
1005 x := isin("-grep",argv); // -grep will be convenient like -ls
1006 if 0 <= x && x+1 < len(argv) { // -grep will be convenient like -ls
1007     if IsRegfile(path){
1008         found := gsh.xGrep(path,argv[x+1:])
1009         if 0 < found {
1010             foundv := gsh.CmdCurrent.FoundFile
1011             if len(foundv) < 10 {
1012                 gsh.CmdCurrent.FoundFile =
1013                     append(gsh.CmdCurrent.FoundFile,path)
1014             }
1015         }
1016     }
1017 }
1018 if !isin("-r0",argv) { // -d 0 in du, -depth n in find
1019     //total.Depth += 1
1020     if (fstat.Mode & S_IFMT) == S_IFLNK {
1021         continue
1022     }
1023     if dstat.Rdev != fstat.Rdev {
1024         fmt.Printf("--I-- don't follow differnet device %v(%v) %v(%v)\n",
1025             dir,dstat.Rdev,path,fstat.Rdev)
1026     }
1027     if (fstat.Mode & S_IFMT) == S_IFDIR {
1028         total = gsh.xxFind(depth+1,total,path,npats,argv)
1029     }
1030 }
1031 }
1032 }
1033 }
1034 return total
1035 func (gsh*GshContext)xxFind(depth int,total *fileSum,dir string,npatsv[]string,argvv[]string)(*fileSum){
1036     nols := isin("-grep",argvv)
1037     dirfile,oerr := os.Openfile(dir,os.O_RDONLY,0)
1038     if oerr == nil {
1039         //fmt.Printf("--I-- %v(%v)[%d]\n",dir,dirfile,dirfile.Fd())
1040         defer dirfile.Close()
1041     }else{
1042     }
1043     prev := *total
1044     var dstat syscall.Stat_t
1045     staterr := syscall.Lstat(dir,&dstat) // should be fstat
1046     if staterr != nil {
1047         if !isin("-w",argvv){ fmt.Printf("ufind: %v\n",staterr) }
1048         return total
1049     }
1050     //filev,err := ioutil.ReadDir(dir)
1051     //_,err := ioutil.ReadDir(dir) // ReadDir() heavy and bad for huge directory
1052     if err != nil {
1053         if !isin("-w",argvv){ fmt.Printf("ufind: %v\n",err) }
1054         return total
1055     }
1056     /*
1057     if depth == 0 {
1058         total = cumFileInfo(total,dir,staterr,dstat,argvv,true)
1059         if !nols && !isin("-s",argvv) && (!isin("-du",argvv) || isin("-a",argvv)) {
1060             showFileInfo(dir,argvv)
1061         }
1062     }
1063     // it is not a directory, just scan it and finish
1064     for ei := 0 ; ; ei++ {
1065         entv,rdrerr := dirfile.Readdirnames(8*1024)
1066         if len(entv) == 0 || rdrerr != nil {
1067             //if rdrerr != nil { fmt.Printf("[%d] len=%d (%v)\n",ei,len(entv),rdrerr) }
1068             break
1069         }
1070         if 0 < ei {
1071             fmt.Printf("--I-- xxFind[%d] %d large-dir: %s\n",ei,len(entv),dir)
1072         }
1073         total = gsh.xxFindEnvt(depth,total,dir,dstat,ei,entv,npatsv,argvv)
1074     }
1075     if isin("-du",argvv) {
1076         // if in "du" mode
1077         fmt.Printf("%d\t%s\n", (total.Blocks-prev.Blocks)/2,dir)
1078     }
1079     return total
1080 }
1081 */
1082 // {ufind|fu|ls} [Files] [-- Names] [-- Expressions]
1083 //   Files is "." by default
1084 //   Names is "*" by default
1085 //   Expressions is "print" by default for "ufind", or -du for "fu" command
1086 func (gsh*GshContext)xFind(argvv[]string){
1087     if 0 < len(argvv) && strBegins(argvv[0],"?"){
1088         showFound(gsh,argvv)
1089         return
1090     }
1091     if isin("-cksum",argvv) || isin("-sum",argvv) {
1092         gsh.lastCheckSum = CheckSum{}
1093         if isin("-sum",argvv) && isin("-add",argvv) {
1094             gsh.lastCheckSum.SumType |= SUM_SUM64
1095         }else{
1096             if isin("-sum",argvv) && isin("-size",argvv) {
1097                 gsh.lastCheckSum.SumType |= SUM_SIZE
1098             }else{
1099                 if isin("-sum",argvv) && isin("-bsd",argvv) {
1100                     gsh.lastCheckSum.SumType |= SUM_SUM16_BSD
1101                 }else{
1102                     if isin("-sum",argvv) && isin("-sysv",argvv) {
1103                         gsh.lastCheckSum.SumType |= SUM_SUM16_SYSV
1104                     }else{
1105                         if isin("-sum",argvv) {
1106                             gsh.lastCheckSum.SumType |= SUM_SUM64
1107                         }
1108                         if isin("-unix",argvv) {
1109                             gsh.lastCheckSum.SumType |= SUM_UNIXFILE
1110                             gsh.lastCheckSum.Crc32Table = *crc32.MakeTable(CRC32UNIX)
1111                         }
1112                         if isin("-ieee",argvv){
1113                             gsh.lastCheckSum.SumType |= SUM_CRCIEEE
1114                             gsh.lastCheckSum.Crc32Table = *crc32.MakeTable(CRC32IEEE)
1115                         }
1116                         gsh.lastCheckSum.RusgAtStart = Getrusagev()
1117                     }
1118                     var total = fileSum{}
1119                     npats := []string{}
1120                     for _v := range argvv {
1121                         npats := append(npats,_v)
1122                     }
1123                     for _v := range argvv {
1124                         npats := append(npats,_v)
1125                     }
1126                     npats := npats[1:]
1127                     for _,v := range npats {
1128                         npatv := strings.Split(v," ")
1129                         if len(npatv) > 1 {
1130                             npatv[1] = strings.TrimSpace(npats[1])
1131                         }
1132                         npats = append(npats,npatv...)
1133                     }
1134                     npats := npats[1:]
1135                     for _,v := range npats {
1136                         npatv := strings.Split(v," ")
1137                         if len(npatv) > 1 {
1138                             npatv[1] = strings.TrimSpace(npats[1])
1139                         }
1140                         npats = append(npats,npatv...)
1141                     }
1142                     npats := npats[1:]
1143                     for _,v := range npats {
1144                         npatv := strings.Split(v," ")
1145                         if len(npatv) > 1 {
1146                             npatv[1] = strings.TrimSpace(npats[1])
1147                         }
1148                         npats = append(npats,npatv...)
1149                     }
1150                     npats := npats[1:]
1151                     for _,v := range npats {
1152                         npatv := strings.Split(v," ")
1153                         if len(npatv) > 1 {
1154                             npatv[1] = strings.TrimSpace(npats[1])
1155                         }
1156                         npats = append(npats,npatv...)
1157                     }
1158                     npats := npats[1:]
1159                     for _,v := range npats {
1160                         npatv := strings.Split(v," ")
1161                         if len(npatv) > 1 {
1162                             npatv[1] = strings.TrimSpace(npats[1])
1163                         }
1164                         npats = append(npats,npatv...)
1165                     }
1166                     npats := npats[1:]
1167                     for _,v := range npats {
1168                         npatv := strings.Split(v," ")
1169                         if len(npatv) > 1 {
1170                             npatv[1] = strings.TrimSpace(npats[1])
1171                         }
1172                         npats = append(npats,npatv...)
1173                     }
1174                     npats := npats[1:]
1175                     for _,v := range npats {
1176                         npatv := strings.Split(v," ")
1177                         if len(npatv) > 1 {
1178                             npatv[1] = strings.TrimSpace(npats[1])
1179                         }
1180                         npats = append(npats,npatv...)
1181                     }
1182                     npats := npats[1:]
1183                     for _,v := range npats {
1184                         npatv := strings.Split(v," ")
1185                         if len(npatv) > 1 {
1186                             npatv[1] = strings.TrimSpace(npats[1])
1187                         }
1188                         npats = append(npats,npatv...)
1189                     }
1190                     npats := npats[1:]
1191                     for _,v := range npats {
1192                         npatv := strings.Split(v," ")
1193                         if len(npatv) > 1 {
1194                             npatv[1] = strings.TrimSpace(npats[1])
1195                         }
1196                         npats = append(npats,npatv...)
1197                     }
1198                     npats := npats[1:]
1199                     for _,v := range npats {
1200                         npatv := strings.Split(v," ")
1201                         if len(npatv) > 1 {
1202                             npatv[1] = strings.TrimSpace(npats[1])
1203                         }
1204                         npats = append(npats,npatv...)
1205                     }
1206                     npats := npats[1:]
1207                     for _,v := range npats {
1208                         npatv := strings.Split(v," ")
1209                         if len(npatv) > 1 {
1210                             npatv[1] = strings.TrimSpace(npats[1])
1211                         }
1212                         npats = append(npats,npatv...)
1213                     }
1214                     npats := npats[1:]
1215                     for _,v := range npats {
1216                         npatv := strings.Split(v," ")
1217                         if len(npatv) > 1 {
1218                             npatv[1] = strings.TrimSpace(npats[1])
1219                         }
1220                         npats = append(npats,npatv...)
1221                     }
1222                     npats := npats[1:]
1223                     for _,v := range npats {
1224                         npatv := strings.Split(v," ")
1225                         if len(npatv) > 1 {
1226                             npatv[1] = strings.TrimSpace(npats[1])
1227                         }
1228                         npats = append(npats,npatv...)
1229                     }
1230                     npats := npats[1:]
1231                     for _,v := range npats {
1232                         npatv := strings.Split(v," ")
1233                         if len(npatv) > 1 {
1234                             npatv[1] = strings.TrimSpace(npats[1])
1235                         }
1236                         npats = append(npats,npatv...)
1237                     }
1238                     npats := npats[1:]
1239                     for _,v := range npats {
1240                         npatv := strings.Split(v," ")
1241                         if len(npatv) > 1 {
1242                             npatv[1] = strings.TrimSpace(npats[1])
1243                         }
1244                         npats = append(npats,npatv...)
1245                     }
1246                     npats := npats[1:]
1247                     for _,v := range npats {
1248                         npatv := strings.Split(v," ")
1249                         if len(npatv) > 1 {
1250                             npatv[1] = strings.TrimSpace(npats[1])
1251                         }
1252                         npats = append(npats,npatv...)
1253                     }
1254                     npats := npats[1:]
1255                     for _,v := range npats {
1256                         npatv := strings.Split(v," ")
1257                         if len(npatv) > 1 {
1258                             npatv[1] = strings.TrimSpace(npats[1])
1259                         }
1260                         npats = append(npats,npatv...)
1261                     }
1262                     npats := npats[1:]
1263                     for _,v := range npats {
1264                         npatv := strings.Split(v," ")
1265                         if len(npatv) > 1 {
1266                             npatv[1] = strings.TrimSpace(npats[1])
1267                         }
1268                         npats = append(npats,npatv...)
1269                     }
1270                     npats := npats[1:]
1271                     for _,v := range npats {
1272                         npatv := strings.Split(v," ")
1273                         if len(npatv) > 1 {
1274                             npatv[1] = strings.TrimSpace(npats[1])
1275                         }
1276                         npats = append(npats,npatv...)
1277                     }
1278                     npats := npats[1:]
1279                     for _,v := range npats {
1280                         npatv := strings.Split(v," ")
1281                         if len(npatv) > 1 {
1282                             npatv[1] = strings.TrimSpace(npats[1])
1283                         }
1284                         npats = append(npats,npatv...)
1285                     }
1286                     npats := npats[1:]
1287                     for _,v := range npats {
1288                         npatv := strings.Split(v," ")
1289                         if len(npatv) > 1 {
1290                             npatv[1] = strings.TrimSpace(npats[1])
1291                         }
1292                         npats = append(npats,npatv...)
1293                     }
1294                     npats := npats[1:]
1295                     for _,v := range npats {
1296                         npatv := strings.Split(v," ")
1297                         if len(npatv) > 1 {
1298                             npatv[1] = strings.TrimSpace(npats[1])
1299                         }
1300                         npats = append(npats,npatv...)
1301                     }
1302                     npats := npats[1:]
1303                     for _,v := range npats {
1304                         npatv := strings.Split(v," ")
1305                         if len(npatv) > 1 {
1306                             npatv[1] = strings.TrimSpace(npats[1])
1307                         }
1308                         npats = append(npats,npatv...)
1309                     }
1310                     npats := npats[1:]
1311                     for _,v := range npats {
1312                         npatv := strings.Split(v," ")
1313                         if len(npatv) > 1 {
1314                             npatv[1] = strings.TrimSpace(npats[1])
1315                         }
1316                         npats = append(npats,npatv...)
1317                     }
1318                     npats := npats[1:]
1319                     for _,v := range npats {
1320                         npatv := strings.Split(v," ")
1321                         if len(npatv) > 1 {
1322                             npatv[1] = strings.TrimSpace(npats[1])
1323                         }
1324                         npats = append(npats,npatv...)
1325                     }
1326                     npats := npats[1:]
1327                     for _,v := range npats {
1328                         npatv := strings.Split(v," ")
1329                         if len(npatv) > 1 {
1330                             npatv[1] = strings.TrimSpace(npats[1])
1331                         }
1332                         npats = append(npats,npatv...)
1333                     }
1334                     npats := npats[1:]
1335                     for _,v := range npats {
1336                         npatv := strings.Split(v," ")
1337                         if len(npatv) > 1 {
1338                             npatv[1] = strings.TrimSpace(npats[1])
1339                         }
1340                         npats = append(npats,npatv...)
1341                     }
1342                     npats := npats[1:]
1343                     for _,v := range npats {
1344                         npatv := strings.Split(v," ")
1345                         if len(npatv) > 1 {
1346                             npatv[1] = strings.TrimSpace(npats[1])
1347                         }
1348                         npats = append(npats,npatv...)
1349                     }
1350                     npats := npats[1:]
1351                     for _,v := range npats {
1352                         npatv := strings.Split(v," ")
1353                         if len(npatv) > 1 {
1354                             npatv[1] = strings.TrimSpace(npats[1])
1355                         }
1356                         npats = append(npats,npatv...)
1357                     }
1358                     npats := npats[1:]
1359                     for _,v := range npats {
1360                         npatv := strings.Split(v," ")
1361                         if len(npatv) > 1 {
1362                             npatv[1] = strings.TrimSpace(npats[1])
1363                         }
1364                         npats = append(npats,npatv...)
1365                     }
1366                     npats := npats[1:]
1367                     for _,v := range npats {
1368                         npatv := strings.Split(v," ")
1369                         if len(npatv) > 1 {
1370                             npatv[1] = strings.TrimSpace(npats[1])
1371                         }
1372                         npats = append(npats,npatv...)
1373                     }
1374                     npats := npats[1:]
1375                     for _,v := range npats {
1376                         npatv := strings.Split(v," ")
1377                         if len(npatv) > 1 {
1378                             npatv[1] = strings.TrimSpace(npats[1])
1379                         }
1380                         npats = append(npats,npatv...)
1381                     }
1382                     npats := npats[1:]
1383                     for _,v := range npats {
1384                         npatv := strings.Split(v," ")
1385                         if len(npatv) > 1 {
1386                             npatv[1] = strings.TrimSpace(npats[1])
1387                         }
1388                         npats = append(npats,npatv...)
1389                     }
1390                     npats := npats[1:]
1391                     for _,v := range npats {
1392                         npatv := strings.Split(v," ")
1393                         if len(npatv) > 1 {
1394                             npatv[1] = strings.TrimSpace(npats[1])
1395                         }
1396                         npats = append(npats,npatv...)
1397                     }
1398                     npats := npats[1:]
1399                     for _,v := range npats {
1400                         npatv := strings.Split(v," ")
1401                         if len(npatv) > 1 {
1402                             npatv[1] = strings.TrimSpace(npats[1])
1403                         }
1404                         npats = append(npats,npatv...)
1405                     }
1406                     npats := npats[1:]
1407                     for _,v := range npats {
1408                         npatv := strings.Split(v," ")
1409                         if len(npatv) > 1 {
1410                             npatv[1] = strings.TrimSpace(npats[1])
1411                         }
1412                         npats = append(npats,npatv...)
1413                     }
1414                     npats := npats[1:]
1415                     for _,v := range npats {
1416                         npatv := strings.Split(v," ")
1417                         if len(npatv) > 1 {
1418                             npatv[1] = strings.TrimSpace(npats[1])
1419                         }
1420                         npats = append(npats,npatv...)
1421                     }
1422                     npats := npats[1:]
1423                     for _,v := range npats {
1424                         npatv := strings.Split(v," ")
1425                         if len(npatv) > 1 {
1426                             npatv[1] = strings.TrimSpace(npats[1])
1427                         }
1428                         npats = append(npats,npatv...)
1429                     }
1430                     npats := npats[1:]
1431                     for _,v := range npats {
1432                         npatv := strings.Split(v," ")
1433                         if len(npatv) > 1 {
1434                             npatv[1] = strings.TrimSpace(npats[1])
1435                         }
1436                         npats = append(npats,npatv...)
1437                     }
1438                     npats := npats[1:]
1439                     for _,v := range npats {
1440                         npatv := strings.Split(v," ")
1441                         if len(npatv) > 1 {
1442                             npatv[1] = strings.TrimSpace(npats[1])
1443                         }
1444                         npats = append(npats,npatv...)
1445                     }
1446                     npats := npats[1:]
1447                     for _,v := range npats {
1448                         npatv := strings.Split(v," ")
1449                         if len(npatv) > 1 {
1450                             npatv[1] = strings.TrimSpace(npats[1])
1451                         }
1452                         npats = append(npats,npatv...)
1453                     }
1454                     npats := npats[1:]
1455                     for _,v := range npats {
1456                         npatv := strings.Split(v," ")
1457                         if len(npatv) > 1 {
1458                             npatv[1] = strings.TrimSpace(npats[1])
1459                         }
1460                         npats = append(npats,npatv...)
1461                     }
1462                     npats := npats[1:]
1463                     for _,v := range npats {
1464                         npatv := strings.Split(v," ")
1465                         if len(npatv) > 1 {
1466                             npatv[1] = strings.TrimSpace(npats[1])
1467                         }
1468                         npats = append(npats,npatv...)
1469                     }
1470                     npats := npats[1:]
1471                     for _,v := range npats {
1472                         npatv := strings.Split(v," ")
1473                         if len(npatv) > 1 {
1474                             npatv[1] = strings.TrimSpace(npats[1])
1475                         }
1476                         npats = append(npats,npatv...)
1477                     }
1478                     npats := npats[1:]
1479                     for _,v := range npats {
1480                         npatv := strings.Split(v," ")
1481                         if len(npatv) > 1 {
1482                             npatv[1] = strings.TrimSpace(npats[1])
1483                         }
1484                         npats = append(npats,npatv...)
1485                     }
1486                     npats := npats[1:]
1487                     for _,v := range npats {
1488                         npatv := strings.Split(v," ")
1489                         if len(npatv) > 1 {
1490                             npatv[1] = strings.TrimSpace(npats[1])
1491                         }
1492                         npats = append(npats,npatv...)
1493                     }
1494                     npats := npats[1:]
1495                     for _,v := range npats {
1496                         npatv := strings.Split(v," ")
1497                         if len(npatv) > 1 {
1498                             npatv[1] = strings.TrimSpace(npats[1])
1499                         }
1500                         npats = append(npats,npatv...)
1501                     }
1502                     npats := npats[1:]
1503                     for _,v := range npats {
1504                         npatv := strings.Split(v," ")
1505                         if len(npatv) > 1 {
1506                             npatv[1] = strings.TrimSpace(npats[1])
1507                         }
1508                         npats = append(npats,npatv...)
1509                     }
1510                     npats := npats[1:]
1511                     for _,v := range npats {
1512                         npatv := strings.Split(v," ")
1513                         if len(npatv) > 1 {
1514                             npatv[1] = strings.TrimSpace(npats[1])
1515                         }
1516                         npats = append(npats,npatv...)
1517                     }
1518                     npats := npats[1:]
1519                     for _,v := range npats {
1520                         npatv := strings.Split(v," ")
1521                         if len(npatv) > 1 {
1522                             npatv[1] = strings.TrimSpace(npats[1])
1523                         }
1524                         npats = append(npats,npatv...)
1525                     }
1526                     npats := npats[1:]
1527                     for _,v := range npats {
1528                         npatv := strings.Split(v," ")
1529                         if len(npatv) > 1 {
1530                             npatv[1] = strings.TrimSpace(npats[1])
1531                         }
1532                         npats = append(npats,npatv...)
1533                     }
1534                     npats := npats[1:]
1535                     for _,v := range npats {
1536                         npatv := strings.Split(v," ")
1537                         if len(npatv) > 1 {
1538                             npatv[1] = strings.TrimSpace(npats[1])
1539                         }
1540                         npats = append(npats,npatv...)
1541                     }
1542                     npats := npats[1:]
1543                     for _,v := range npats {
1544                         npatv := strings.Split(v," ")
1545                         if len(npatv) > 1 {
1546                             npatv[1] = strings.TrimSpace(npats[1])
1547                         }
1548                         npats = append(npats,npatv...)
1549                     }
1550                     npats := npats[1:]
1551                     for _,v := range npats {
1552                         npatv := strings.Split(v," ")
1553                         if len(npatv) > 1 {
1554                             npatv[1] = strings.TrimSpace(npats[1])
1555                         }
1556                         npats = append(npats,npatv...)
1557                     }
1558                     npats := npats[1:]
1559                     for _,v := range npats {
1560                         npatv := strings.Split(v," ")
1561                         if len(npatv) > 1 {
1562                             npatv[1] = strings.TrimSpace(npats[1])
1563                         }
1564                         npats = append(npats,npatv...)
1565                     }
1566                     npats := npats[1:]
1567                     for _,v := range npats {
1568                         npatv := strings.Split(v," ")
1569                         if len(npatv) > 1 {
1570                             npatv[1] = strings.TrimSpace(npats[1])
1571                         }
1572                         npats = append(npats,npatv...)
1573                     }
1574                     npats := npats[1:]
1575                     for _,v := range npats {
1576                         npatv := strings.Split(v," ")
1577                         if len(npatv) > 1 {
1578                             npatv[1] = strings.TrimSpace(npats[1])
1579                         }
1580                         npats = append(npats,npatv...)
1581                     }
1582                     npats := npats[1:]
1583                     for _,v := range npats {
1584                         npatv := strings.Split(v," ")
1585                         if len(npatv) > 1 {
1586                             npatv[1] = strings.TrimSpace(npats[1])
1587                         }
1588                         npats = append(npats,npatv...)
1589                     }
1590                     npats := npats[1:]
1591                     for _,v := range npats {
1592                         npatv := strings.Split(v," ")
1593                         if len(npatv) > 1 {
1594                             npatv[1] = strings.TrimSpace(npats[1])
1595                         }
1596                         npats = append(npats,npatv...)
1597                     }
1598                     npats := npats[1:]
1599                     for _,v := range npats {
1600                         npatv := strings.Split(v," ")
1601                         if len(npatv) > 1 {
1602                             npatv[1] = strings.TrimSpace(npats[1])
1603                         }
1604                         npats = append(npats,npatv...)
1605                     }
1606                     npats := npats[1:]
1607                     for _,v := range npats {
1608                         npatv := strings.Split(v," ")
1609                         if len(npatv) > 1 {
1610                             npatv[1] = strings.TrimSpace(npats[1])
1611                         }
1612                         npats = append(npats,npatv...)
1613                     }
1614                     npats := npats[1:]
1615                     for _,v := range npats {
1616                         npatv := strings.Split(v," ")
1617                         if len(npatv) > 1 {
1618                             npatv[1] = strings.TrimSpace(npats[1])
1619                         }
1620                         npats = append(npats,npatv...)
1621                     }
1622                     npats := npats[1:]
1623                     for _,v := range npats {
1624                         npatv := strings.Split(v," ")
1625                         if len(npatv) > 1 {
1626                             npatv[1] = strings.TrimSpace(npats[1])
1627                         }
1628                         npats = append(npats,npatv...)
1629                     }
1630                     npats := npats[1:]
1631                     for _,v := range npats {
1632                         npatv := strings.Split(v," ")
1633                         if len(npatv) > 1 {
1634                             npatv[1] = strings.TrimSpace(npats[1])
1635                         }
1636                         npats = append(npats,npatv...)
1637                     }
1638                     npats := npats[1:]
1639                     for _,v := range npats {
1640                         npatv := strings.Split(v," ")
1641                         if len(npatv) > 1 {
1642                             npatv[1] = strings.TrimSpace(npats[1])
1643                         }
1644                         npats = append(npats,npatv...)
1645                     }
1646                     npats := npats[1:]
1647                     for _,v := range npats {
1648                         npatv := strings.Split(v," ")
1649                         if len(npatv) > 1 {
1650                             npatv[1] = strings.TrimSpace(npats[1])
1651                         }
1652                         npats = append(npats,npatv...)
1653                     }
1654                     npats := npats[1:]
1655                     for _,v := range npats {
1656                         npatv := strings.Split(v," ")
1657                         if len(npatv) > 1 {
1658                             npatv[1] = strings.TrimSpace(npats[1])
1659                         }
1660                         npats = append(npats,npatv...)
1661                     }
1662                     npats := npats[1:]
1663                     for _,v := range npats {
1664                         npatv := strings.Split(v," ")
1665                         if len(npatv) > 1 {
1666                             npatv[1] = strings.TrimSpace(npats[1])
1667                         }
1668                         npats = append(npats,npatv...)
1669                     }
1670                     npats := npats[1:]
1671                     for _,v := range npats {
1672                         npatv := strings.Split(v," ")
1673                         if len(npatv) > 1 {
1674                             npatv[1] = strings.TrimSpace(npats[1])
1675                         }
1676                         npats = append(npats,npatv...)
1677                     }
1678                     npats := npats[1:]
1679                     for _,v := range npats {
1680                         npatv := strings.Split(v," ")
1681                         if len(npatv) > 1 {
1682                             npatv[1] = strings.TrimSpace(npats[1])
1683                         }
1684                         npats = append(npats,npatv...)
1685                     }
1686                     npats := npats[1:]
1687                     for _,v := range npats {
1688                         npatv := strings.Split(v," ")
1689                         if len(npatv) > 1 {
1690                             npatv[1] = strings.TrimSpace(npats[1])
1691                         }
1692                         npats = append(npats,npatv...)
1693                     }
1694                     npats := npats[1:]
1695                     for _,v := range npats {
1696                         npatv := strings.Split(v," ")
1697                         if len(npatv) > 1 {
1698                             npatv[1] = strings.TrimSpace(npats[1])
1699                         }
1700                         npats = append(npats,npatv...)
1701                     }
1702                     npats := npats[1:]
1703                     for _,v := range npats {
1704                         npatv := strings.Split(v," ")
1705                         if len(npatv) > 1 {
1706                             npatv[1] = strings.TrimSpace(npats[1])
1707                         }
1708                         npats = append(npats,npatv...)
1709                     }
1710                     npats := npats[1:]
1711                     for _,v := range npats {
1712                         npatv := strings.Split(v," ")
1713                         if len(npatv) > 1 {
1714                             npatv[1] = strings.TrimSpace(npats[1])
1715                         }
1716                         npats = append(npats,npatv...)
1717                     }
1718                     npats := npats[1:]
1719                     for _,v := range npats {
1720                         npatv := strings.Split(v," ")
1721                         if len(npatv) > 1 {
1722                             npatv[1] = strings.TrimSpace(npats[1])
1723                         }
1724                         npats = append(npats,npatv...)
1725                     }
1726                     npats := npats[1:]
1727                     for _,v := range npats {
1728                         npatv := strings.Split(v," ")
1729                         if len(npatv) > 1 {
1730                             npatv[1] = strings.TrimSpace(npats[1])
1731                         }
1732                         npats = append(npats,npatv...)
1733                     }
1734                     npats := npats[1:]
1735                     for _,v := range npats {
1736                         npatv := strings.Split(v," ")
1737                         if len(npatv) > 1 {
1738                             npatv[1] = strings.TrimSpace(npats[1])
1739                         }
1740                         npats = append(npats,npatv...)
1741                     }
1742                     npats := npats[1:]
1743                     for _,v := range npats {
1744                         npatv := strings.Split(v," ")
1745                         if len(npatv) > 1 {
1746                             npatv[1] = strings.TrimSpace(npats[1])
1747                         }
1748                         npats = append(npats,npatv...)
1749                     }
1750                     npats := npats[1:]
1751                     for _,v := range npats {
1752                         npatv := strings.Split(v," ")
1753                         if len(npatv) > 1 {
1754                             npatv[1] = strings.TrimSpace(npats[1])
1755                         }
1756                         npats = append(npats,npatv...)
1757                     }
1758                     npats := npats[1:]
1759                     for _,v := range npats {
1760                         npatv := strings.Split(v," ")
1761                         if len(npatv) > 1 {
1762                             npatv[1] = strings.TrimSpace(npats[1])
1763                         }
1764                         npats = append(npats,npatv...)
1765                     }
1766                     npats := npats[1:]
1767                     for _,v := range npats {
1768                         npatv := strings.Split(v," ")
1769                         if len(npatv) > 1 {
1770                             npatv[1] = strings.TrimSpace(npats[1])
1771                         }
1772                         npats = append(npats,npatv...)
1773                     }
1774                     npats := npats[1:]
1775                     for _,v := range npats {
1776                         npatv := strings.Split(v," ")
1777                         if len(npatv) > 1 {
1778                             npatv[1] = strings.TrimSpace(npats[1])
1779                         }
1780                         npats = append(npats,npatv...)
1781                     }
1782                     npats := npats[1:]
1783                     for _,v := range npats {
1784                         npatv := strings.Split(v," ")
1785                         if len(npatv) > 1 {
1786                             npatv[1] = strings.TrimSpace(npats[1])
1787                         }
1788                         npats = append(npats,npatv...)
1789                     }
1790                     npats := npats[1:]
1791                     for _,v := range npats {
1792                         npatv := strings.Split(v," ")
1793                         if len(npatv) > 1 {
1794                             npatv[1] = strings.TrimSpace(npats[1])
1795                         }
1796                         npats = append(npats,npatv...)
1797                     }
1798                     npats := npats[1:]
1799                     for _,v := range npats {
1800                         npatv := strings.Split(v," ")
1801                         if len(npatv) > 1 {
1802                             npatv[1] = strings.TrimSpace(npats[1])
1803                         }
1804                         npats = append(npats,npatv...)
1805                     }
1806                     npats := npats[1:]
1807                     for _,v := range npats {
1808                         npatv := strings.Split(v," ")
1809                         if len(npatv) > 1 {
1810                             npatv[1] = strings.TrimSpace(npats[1])
1811                         }
1812                         npats = append(npats,npatv...)
1813                     }
1814                     npats := npats[1:]
1815                     for _,v := range npats {
1816                         npatv := strings.Split(v," ")
1817                         if len(npatv) > 1 {
1818                             npatv[1] = strings.TrimSpace(npats[1])
1819                         }
1820                         npats = append(npats,npatv...)
1821                     }
1822                     npats := npats[1:]
1823                     for _,v := range npats {
1824                         npatv := strings.Split(v," ")
1825                         if len(npatv) > 1 {
1826                             npatv[1] = strings.TrimSpace(npats[1])
1827                         }
1828                         npats = append(npats,npatv...)
1829                     }
1830                     npats := npats[1:]
1831                     for _,v := range npats {
1832                         npatv := strings.Split(v," ")
1833                         if len(npatv) > 1 {
1834                             npatv[1] = strings.TrimSpace(npats[1])
1835                         }
1836                         npats = append(npats,npatv...)
1837                     }
1838                     npats := npats[1:]
1839                     for _,v := range
```

```

1125     if 0 < len(v) && v[0] != '-' {
1126         npats = append(npats,v)
1127     }
1128     if v == "//" { break }
1129     if v == "--" { break }
1130     if v == "-grep" { break }
1131     if v == "-ls" { break }
1132 }
1133 if len(npats) == 0 {
1134     npats = []string{"*"}
1135 }
1136 cwd := "."
1137 // if to be fullpath :: cwd, _ := os.Getwd()
1138 if len(npats) == 0 { npats = []string{"*"} }
1139 fusage := gsh.xxFind(0,&total,cwd,npats,argv)
1140 if gsh.lastCheckSum.SumType != 0 {
1141     var sumi uint64 = 0
1142     sum := &gsh.lastCheckSum
1143     if (sum.SumType & SUM_SIZE) != 0 {
1144         sumi = uint64(sum.Size)
1145     }
1146     if (sum.SumType & SUM_SUM64) != 0 {
1147         sumi = sum.Sum64
1148     }
1149     if (sum.SumType & SUM_SUM16_SYSV) != 0 {
1150         s := uint32(sum.Sum16)
1151         r := (s & 0xFFFF) + ((s & 0xFFFFFFF) >> 16)
1152         s = (r & 0xFFFF) + (r >> 16)
1153         sum.Crc32Val = uint32(s)
1154         sumi = uint64(s)
1155     }
1156     if (sum.SumType & SUM_SUM16_BSD) != 0 {
1157         sum.Crc32Val = uint32(sum.Sum16)
1158         sumi = uint64(sum.Sum16)
1159     }
1160     if (sum.SumType & SUM_UNIXFILE) != 0 {
1161         sum.Crc32Val = byteCRC32end(sum.Crc32Val,uint64(sum.Size))
1162         sumi = uint64(byteCRC32end(sum.Crc32Val,uint64(sum.Size)))
1163     }
1164     if 1 < sum.Files {
1165         fmt.Printf("%v %v files, %v/file\r\n",
1166             sumi,sum.Size,
1167             abssize(sum.Size),sum.Files,
1168             abssize(sum.Size/sum.Files))
1169     }else{
1170         fmt.Printf("%v %v %v\n",
1171             sumi,sum.Size,npats[0])
1172     }
1173 }
1174 if !isin("-grep",argv) {
1175     showFusage("total",fusage)
1176 }
1177 if !isin("-s",argv){
1178     hits := len(gsh.CmdCurrent.FoundFile)
1179     if 0 < hits {
1180         fmt.Printf("--I-- %d files hits // can be refered with !df\n",
1181             hits,len(gsh.CommandHistory))
1182     }
1183 }
1184 if gsh.lastCheckSum.SumType != 0 {
1185     if isin("-ru",argv) {
1186         sum := &gsh.lastCheckSum
1187         sum.Done = time.Now()
1188         gsh.lastCheckSum.RusgAtEnd = Getrusagev()
1189         elps := sum.Done.Sub(sum.Start)
1190         fmt.Printf("--cksum-size: %v (%v) / %v files, %v/file\r\n",
1191             sum.Size,abssize(sum.Size),sum.Files,abssize(sum.Size/sum.Files))
1192         nanos := int64(elps)
1193         fmt.Printf("--cksum-time: %v/total, %v/file, %.1f files/s, %v\r\n",
1194             abbtme(nanos),
1195             abbtme(nanos/sum.Files),
1196             (float64(sum.Files)*1000000000.0)/float64(nanos),
1197             abbspeed(sum.Size,nanos))
1198         diff := RusageSub(sum.RusgAtEnd,sum.RusgAtStart)
1199         fmt.Printf("--cksum-rusg: %v\n",sRusagef("",argv,diff))
1200     }
1201 }
1202 return
1203 }
1204
1205 func showFiles(files[]string){
1206     sp := ""
1207     for i,file := range files {
1208         if 0 < i { sp = " " } else { sp = "" }
1209         fmt.Printf(sp+"%s",escapeWhiteSP(file))
1210     }
1211 }
1212 func showFound(gshCtx *GshContext, argv[]string){
1213     for i,v := range gshCtx.CommandHistory {
1214         if 0 < len(v.FoundFile) {
1215             fmt.Printf("%d (%d) ",i,len(v.FoundFile))
1216             if isin("-ls",argv){
1217                 fmt.Println("\n")
1218                 for _file := range v.FoundFile {
1219                     fmt.Printf("%") //sub number?
1220                     showFileInfo(file,argv)
1221                 }
1222             }else{
1223                 showFiles(v.FoundFile)
1224                 fmt.Println("\n")
1225             }
1226         }
1227     }
1228 }
1229
1230 func showMatchFile(filev []os.FileInfo, npat,dir string, argv[]string)(string,bool){
1231     fname := ""
1232     found := false
1233     for _v := range filev {
1234         match, _ := filepath.Match(npat,(v.Name()))
1235         if match {
1236             fname = v.Name()
1237             found = true
1238             //fmt.Printf("[%d] %s\n",i,v.Name())
1239             showIfExecutable(fname,dir,argv)
1240         }
1241     }
1242     return fname,found
1243 }
1244 func showIfExecutable(name,dir string,argv[]string)(ffullpath string,ffound bool){
1245     var fullpath string
1246     if strBegins(name,DIRSEP){
1247         fullpath = name
1248     }else{
1249         fullpath = dir + DIRSEP + name

```

```

1250 }
1251 fi, err := os.Stat(fullpath)
1252 if err != nil {
1253     fullpath = dir + DIRSEP + name + ".go"
1254     fi, err = os.Stat(fullpath)
1255 }
1256 if err == nil {
1257     fm := fi.Mode()
1258     if fm.IsRegular() {
1259         // R_OK=4, W_OK=2, X_OK=1, F_OK=0
1260         if syscall.Access(fullpath, 5) == nil {
1261             ffullpath = fullpath
1262             ffound = true
1263             if !isin("-s", argv) {
1264                 showFileInfo(fullpath, argv)
1265             }
1266         }
1267     }
1268 }
1269 return ffullpath, ffound
1270 }
1271 func which(list string, argv []string) (fullpathv []string, itis bool){
1272 if len(argv) <= 1 {
1273     fmt.Printf("Usage: which comand [-s] [-a] [-ls]\n")
1274     return []string{}, false
1275 }
1276 path := argv[1]
1277 if strBegins(path, "/") {
1278     // should check if executable?
1279     _exOK := showIfExecutable(path, "/", argv)
1280     fmt.Printf("--D-- %s\n", path, exOK)
1281     return []string{path}, exOK
1282 }
1283 pathenv, efound := os.LookupEnv(list)
1284 if !efound {
1285     fmt.Printf("--E-- which: no \"%s\" environment\n", list)
1286     return []string{}, false
1287 }
1288 showall := isin("-a", argv) || 0 <= strings.Index(path, "*")
1289 dirv := strings.Split(pathenv, PATHSEP)
1290 ffound := false
1291 ffullpath := path
1292 for _, dir := range dirv {
1293     if 0 <= strings.Index(path, "*") { // by wild-card
1294         list, _ := ioutil.ReadDir(dir)
1295         ffullpath, ffound = showMatchFile(list, path, dir, argv)
1296     }else{
1297         ffullpath, ffound = showIfExecutable(path, dir, argv)
1298     }
1299     //if ffound && !showall {
1300     if ffound && !showall {
1301         break;
1302     }
1303 }
1304 return []string{ffullpath}, ffound
1305 }
1306
1307 func stripLeadingWSParg(argv[]string)([]string){
1308 for ; 0 < len(argv); {
1309     if len(argv[0]) == 0 {
1310         argv = argv[1:]
1311     }else{
1312         break
1313     }
1314 }
1315 return argv
1316 }
1317 func xEval(argv []string, nlend bool){
1318 argv = stripLeadingWSParg(argv)
1319 if len(argv) == 0 {
1320     fmt.Printf("eval [%s] [Go-expression]\n")
1321     return
1322 }
1323 pfmt := "%v"
1324 if argv[0][0] == '%' {
1325     pfmt = argv[0]
1326     argv = argv[1:]
1327 }
1328 if len(argv) == 0 {
1329     return
1330 }
1331 gocode := strings.Join(argv, " ");
1332 //fmt.Printf("eval [%v] [%v]\n", pfmt, gocode)
1333 fset := token.NewFileSet()
1334 rval, _ := types.Eval(fset, nil, token.NoPos, gocode)
1335 fmt.Printf(pfmt, rval.Value)
1336 if nlend { fmt.Printf("\n") }
1337 }
1338
1339 func getval(name string) (found bool, val int) {
1340 /* should expand the name here */
1341 if name == "gsh.pid" {
1342     return true, os.Getpid()
1343 }else{
1344     if name == "gsh.ppid" {
1345         return true, os.Getppid()
1346     }
1347     return false, 0
1348 }
1349
1350 func echo(argv []string, nlend bool){
1351 for ai := 1; ai < len(argv); ai++ {
1352     if 1 < ai {
1353         fmt.Printf(" ");
1354     }
1355     arg := argv[ai]
1356     found, val := getval(arg)
1357     if found {
1358         fmt.Printf("%d", val)
1359     }else{
1360         fmt.Printf("%s", arg)
1361     }
1362 }
1363 if nlend {
1364     fmt.Printf("\n");
1365 }
1366 }
1367
1368 func resfile() string {
1369     return "gsh.tmp"
1370 }
1371 //var resF *File
1372 func resmap() {
1373     _, err := os.OpenFile(resfile(), os.O_RDWR|os.O_CREATE, os.ModeAppend)
1374     // https://developpaper.com/solution-to-golang-bad-file-descriptor-problem/

```

```

1375     , err := os.OpenFile(resfile(), os.O_RDWR|os.O_CREATE, 0600)
1376     if err != nil {
1377         fmt.Printf("refF could not open: %s\n",err)
1378     }else{
1379         fmt.Printf("refF opened\n")
1380     }
1381 }
1382
1383 // @02020-0821
1384 func gshScanArg(str string,strip int)(argv []string){
1385     var si = 0
1386     var sb = 0
1387     var inBracket = 0
1388     var arg1 = make([]byte,LINESIZE)
1389     var ax = 0
1390     debug := false
1391
1392     for ; si < len(str); si++ {
1393         if str[si] != ' ' {
1394             break
1395         }
1396     }
1397     sb = si
1398     for ; si < len(str); si++ {
1399         if sb <= si {
1400             if debug {
1401                 fmt.Printf("--Da- +$d %2d-%2d $s ... $s\n",
1402                     inBracket,sb,si,arg1[0:ax],str[si:])
1403             }
1404             ch := str[si]
1405             if ch == '{' {
1406                 inBracket += 1
1407                 if 0 < strip && inBracket <= strip {
1408                     //fmt.Printf("stripLEV $d <= $d?\n",inBracket,strip)
1409                     continue
1410                 }
1411             }
1412             if 0 < inBracket {
1413                 if ch == ')' {
1414                     inBracket -= 1
1415                     if 0 < strip && inBracket < strip {
1416                         //fmt.Printf("stripLEV $d < $d?\n",inBracket,strip)
1417                         continue
1418                     }
1419                 }
1420                 arg1[ax] = ch
1421                 ax += 1
1422                 continue
1423             }
1424             if str[si] == ' ' {
1425                 argv = append(argv,string(arg1[0:ax]))
1426                 if debug {
1427                     fmt.Printf("--Da- [%v][%v-%v] $s ... $s\n",
1428                         -1+len(argv),sb,si,str[sb:si],string(str[si:]))
1429                 }
1430                 sb = si+1
1431                 ax = 0
1432                 continue
1433             }
1434             arg1[ax] = ch
1435             ax += 1
1436         }
1437     }
1438     if sb < si {
1439         argv = append(argv,string(arg1[0:ax]))
1440         if debug {
1441             fmt.Printf("--Da- [%v][%v-%v] $s ... $s\n",
1442                 -1+len(argv),sb,si,string(arg1[0:ax]),string(str[si:]))
1443         }
1444     }
1445     if debug {
1446         fmt.Printf("--Da- %d [%s] => [%d]$v\n",strip,str,len(argv),argv)
1447     }
1448 }
1449 }
1450
1451 // should get stderr (into tmpfile ?) and return
1452 func (gsh*GshContext)Popen(name,mode string)(pin*os.File,pout*os.File,err bool){
1453     var pv = []int{-1,-1}
1454     syscall.Pipe(pv)
1455
1456     xarg := gshScanArg(name,1)
1457     name = strings.Join(xarg," ")
1458
1459     pin = os.NewFile(uintptr(pv[0]),"StdoutOf-{"+name+"}")
1460     pout = os.NewFile(uintptr(pv[1]),"StdinOf-{"+name+"}")
1461     ffix := 0
1462     dir := "?"
1463     if mode == "r" {
1464         dir = "<"
1465         ffix = 1 // read from the stdout of the process
1466     }else{
1467         dir = ">"
1468         ffix = 0 // write to the stdin of the process
1469     }
1470     gshPA := gsh.gshPA
1471     savfd := gshPA.Files[ffix]
1472
1473     var fd uintptr = 0
1474     if mode == "r" {
1475         fd = pout.Fd()
1476         gshPA.Files[ffix] = pout.Fd()
1477     }else{
1478         fd = pin.Fd()
1479         gshPA.Files[ffix] = pin.Fd()
1480     }
1481     // should do this by Goroutine?
1482     if false {
1483         fmt.Printf("--Ip- Opened fd[%v] $s $v\n",fd,dir,name)
1484         fmt.Printf("--REDL [$_,$d,$d]-[$d,$d,$d]\n",
1485             os.Stdin.Fd(),os.Stdout.Fd(),os.Stderr.Fd(),
1486             pin.Fd(),pout.Fd(),pout.Fd())
1487     }
1488     savi := os.Stdin
1489     save := os.Stdout
1490     save := os.Stderr
1491     os.Stdin = pin
1492     os.Stdout = pout
1493     os.Stderr = pout
1494     gsh.BackGround = true
1495     gsh.gshellh(name)
1496     gsh.BackGround = false
1497     os.Stdin = savi
1498     os.Stdout = save
1499     os.Stderr = save

```

```

1500
1501     gshPA.Files[fdix] = savfd
1502     return pin,pout,false
1503 }
1504
1505 // <a name="ex-commands">External commands</a>
1506 func (gsh*GshContext)execCommand(exec bool, argv []string) (notf bool,exit bool) {
1507     if gsh.CmdTrace { fmt.Printf("--I-- excommand[%v](%v)\n",exec,argv) }
1508
1509     gshPA := gsh.gshPA
1510     fullpathv, itis := which("PATH",[]string{"which",argv[0],"-s"})
1511     if itis == false {
1512         return true,false
1513     }
1514     fullpath := fullpathv[0]
1515     argv = unescapeWhiteSPV(argv)
1516     if 0 < strings.Index(fullpath,".go") {
1517         nargv := argv // []string{}
1518         gofullpathv, itis := which("PATH",[]string{"which","go","-s"})
1519         if itis == false {
1520             fmt.Println("--F-- Go not found\n")
1521             return false,true
1522         }
1523         gofullpath := gofullpathv[0]
1524         nargv = []string{ gofullpath, "run", fullpath }
1525         fmt.Printf("--I-- %s %s\n",gofullpath,
1526                         nargv[0],nargv[1],nargv[2])
1527         if exec {
1528             syscall.Exec(gofullpath,nargv,os.Environ())
1529         }else{
1530             pid, _ := syscall.ForkExec(gofullpath,nargv,&gshPA)
1531             if gsh.BackGround {
1532                 fmt.Fprintf(stderr,"--Ip- in Background pid[%d]%d(%v)\n",pid,len(argv),nargv)
1533                 gsh.BackGroundJobs = append(gsh.BackGroundJobs,pid)
1534             }else{
1535                 rusage := syscall.Rusage {}
1536                 syscall.Wait4(pid,nil,0,&rusage)
1537                 gsh.LastRusage = rusage
1538                 gsh.CmdCurrent.Rusagev[1] = rusage
1539             }
1540         }
1541     }else{
1542         if exec {
1543             syscall.Exec(fullpath,argv,os.Environ())
1544         }else{
1545             pid, _ := syscall.ForkExec(fullpath,argv,&gshPA)
1546             //fmt.Printf("[%d]\n",pid); // '&' to be background
1547             if gsh.BackGround {
1548                 fmt.Fprintf(stderr,"--Ip- in Background pid[%d]%d(%v)\n",pid,len(argv),nargv)
1549                 gsh.BackGroundJobs = append(gsh.BackGroundJobs,pid)
1550             }else{
1551                 rusage := syscall.Rusage {}
1552                 syscall.Wait4(pid,nil,0,&rusage)
1553                 gsh.LastRusage = rusage
1554                 gsh.CmdCurrent.Rusagev[1] = rusage
1555             }
1556         }
1557     }
1558     return false,false
1559 }
1560
1561 // <a name="builtin">Builtin Commands</a>
1562 func (gshCtx *GshContext) sleep(argv []string) {
1563     if len(argv) < 2 {
1564         fmt.Printf("Sleep 100ms, 100us, 100ns, ...\n")
1565         return
1566     }
1567     duration := argv[1];
1568     d, err := time.ParseDuration(duration)
1569     if err != nil {
1570         d, err = time.ParseDuration(duration+"s")
1571         if err != nil {
1572             fmt.Printf("duration ? %s (%s)\n",duration,err)
1573             return
1574         }
1575     }
1576     //fmt.Printf("Sleep %v\n",duration)
1577     time.Sleep(d)
1578     if 0 < len(argv[2:]) {
1579         gshCtx.gshellv(argv[2:])
1580     }
1581 }
1582 func (gshCtx *GshContext)repeat(argv []string) {
1583     if len(argv) < 2 {
1584         return
1585     }
1586     start0 := time.Now()
1587     for ri,_ := strconv.Atoi(argv[1]); 0 < ri; ri-- {
1588         if 0 < len(argv[2:]) {
1589             //start := time.Now()
1590             gshCtx.gshellv(argv[2:])
1591             end := time.Now()
1592             elps := end.Sub(start0);
1593             if( 1000000000 < elps ){
1594                 fmt.Printf("(repeat%d %v)\n",ri,elps);
1595             }
1596         }
1597     }
1598 }
1599
1600 func (gshCtx *GshContext)gen(argv []string) {
1601     gshPA := gshCtx.gshPA
1602     if len(argv) < 2 {
1603         fmt.Printf("Usage: %s N\n",argv[0])
1604         return
1605     }
1606     // should br repeated by "repeat" command
1607     count, _ := strconv.Atoi(argv[1])
1608     fd := gshPA.Files[1] // Stdout
1609     file := os.NewFile(fd,"internalStdOut")
1610     fmt.Printf("--I-- Gen. Count=%d to [%d]\n",count,file.Fd())
1611     //buf := []byte{}
1612     outdata := "0123 5678 0123 5678 0123 5678 0123 5678\r"
1613     for gi := 0; gi < count; gi++ {
1614         file.WriteString(outdata)
1615     }
1616     //file.WriteString("\n")
1617     fmt.Printf("\n(%d B)\n",count*len(outdata));
1618     //file.Close()
1619 }
1620
1621 // <a name="rexec">Remote Execution</a> // 2020-0820
1622 func Elapsed(from time.Time)(string){
1623     elps := time.Now().Sub(from)
1624     if 1000000000 < elps {

```

```

1625     return fmt.Sprintf("[%5d.%02ds]", elps/1000000000,(elps%1000000000)/1000000)
1626 }else{
1627 if 1000000 < elps {
1628     return fmt.Sprintf("[%3d.%03dms]", elps/1000000,(elps%1000000)/1000)
1629 }else{
1630     return fmt.Sprintf("[%3d.%03dus]", elps/1000,(elps%1000))
1631 }
1632 }
1633 func abbttime(nanos int64)(string){
1634 if 1000000000 < nanos {
1635     return fmt.Sprintf("%d.%02ds",nanos/1000000000,(nanos%1000000000)/1000000)
1636 }else{
1637     if 1000000 < nanos {
1638         return fmt.Sprintf("%d.%03dms",nanos/1000000,(nanos%1000000)/1000)
1639 }else{
1640         return fmt.Sprintf("%d.%03dus",nanos/1000,(nanos%1000))
1641 }
1642 }
1643 func abssize(size int64)(string){
1644 fsize := float64(size)
1645 if 1024*1024*1024 < size {
1646     return fmt.Sprintf("%.2fGiB",fsize/(1024*1024*1024))
1647 }else{
1648 if 1024*1024 < size {
1649     return fmt.Sprintf("%.3fMiB",fsize/(1024*1024))
1650 }else{
1651     return fmt.Sprintf("%.3fKiB",fsize/1024)
1652 }
1653 }
1654 func absizesize(int64)(string){
1655 fsize := float64(size)
1656 if 1024*1024*1024 < size {
1657     return fmt.Sprintf("%.8.2fGiB",fsize/(1024*1024*1024))
1658 }else{
1659     if 1024*1024 < size {
1660         return fmt.Sprintf("%.8.3fMiB",fsize/(1024*1024))
1661 }else{
1662         return fmt.Sprintf("%.8.3fKiB",fsize/1024)
1663 }
1664 }
1665 func abbspeed(totalB int64,ns int64)(string){
1666 MBs := (float64(totalB)/1000000) / (float64(ns)/1000000000)
1667 if 1000 <= MBs {
1668     return fmt.Sprintf("%.6.3fGB/s",MBs/1000)
1669 }
1670 if 1 <= MBs {
1671     return fmt.Sprintf("%.6.3fMB/s",MBs)
1672 }else{
1673     return fmt.Sprintf("%.6.3fKB/s",MBs*1000)
1674 }
1675 }
1676 func absspeed(totalB int64,ns time.Duration)(string){
1677 MBs := (float64(totalB)/1000000) / (float64(ns)/1000000000)
1678 if 1000 <= MBs {
1679     return fmt.Sprintf("%.6.3fGbps",MBs/1000)
1680 }
1681 if 1 <= MBs {
1682     return fmt.Sprintf("%.6.3fMbps",MBs)
1683 }else{
1684     return fmt.Sprintf("%.6.3fKbps",MBs*1000)
1685 }
1686 }
1687 func fileRelay(what string,in*os.File,out*os.File,size int64,bsiz int)(wcount int64){
1688 Start := time.Now()
1689 buff := make([]byte,bsiz)
1690 var total int64 = 0
1691 var rem int64 = size
1692 nio := 0
1693 Prev := time.Now()
1694 var PrevSize int64 = 0
1695
1696 fmt.Printf(Elapsed(Start)+"--In- X: %s (%v/%v/%v) START\n",
1697 what,absize(total),size,nio)
1698
1699 for i:= 0; ; i++ {
1700     var len = bsiz
1701     if int(rem) < len {
1702         len = int(rem)
1703     }
1704     Now := time.Now()
1705     Elps := Now.Sub(Prev);
1706     if 1000000000 < Now.Sub(Prev) {
1707         fmt.Printf(Elapsed(Start)+"--In- X: %s (%v/%v/%v) %s\n",
1708 what,absize(total),size,nio,
1709 abspeed((total-PrevSize),Elps))
1710     Prev = Now;
1711     PrevSize = total
1712 }
1713 rlen := len
1714 if in != nil {
1715     // should watch the disconnection of out
1716     rcc,err := in.Read(buff[0:rlen])
1717     if err != nil {
1718         fmt.Printf(Elapsed(Start)+"--En- X: %s read(%v,%v)<%v\n",
1719 what,rcc,err,in.Name())
1720         break
1721     }
1722     rlen = rcc
1723     if string(buff[0:10]) == "((SoftEOF "
1724         var ecc int64 = 0
1725         fmt.Sscanf(string(buff),"(SoftEOF %v",&ecc)
1726         fmt.Printf(Elapsed(Start)+"--En- X: %s Recv ((SoftEOF %v))/%v\n",
1727 what,ecc,total)
1728         if ecc == total {
1729             break
1730         }
1731     }
1732 }
1733
1734 wlen := rlen
1735 if out != nil {
1736     wcc,err := out.Write(buff[0:rlen])
1737     if err != nil {
1738         fmt.Printf(Elapsed(Start)+"-En-- X: %s write(%v,%v)>%v\n",
1739 what,wcc,err,out.Name())
1740         break
1741     }
1742     wlen = wcc
1743 }
1744 if wlen < rlen {
1745     fmt.Printf(Elapsed(Start)+"--En- X: %s incomplete write (%v/%v)\n",
1746 what,wlen,rlen)
1747     break;
1748 }

```

```

1750     nio += 1
1751     total += int64(rlen)
1752     rem -= int64(rlen)
1753     if rem <= 0 {
1754         break
1755     }
1756 }
1757 Done := time.Now()
1758 Elps := float64(Done.Sub(Start))/1000000000 //Seconds
1759 TotalMB := float64(total)/1000000 //MB
1760 MBps := TotalMB / Elps
1761 fmt.Printf(Elapsed(Start)+"--In- X: %s (%v/%v/%v) %v %.3fMB/s\n",
1762             what,total,size,nio,absize(total),MBps)
1763 return total
1764 }
1765 func tcpPush(clnt *os.File){
1766     // shrink socket buffer and recover
1767     usleep(100);
1768 }
1769 func (gsh*GshContext)RexecServer(argv[]string){
1770     debug := true
1771     Start0 := time.Now()
1772     Start := Start0
1773 // if local == ":" { local = "0.0.0.0:9999" }
1774     local := "0.0.0.0:9999"
1775
1776     if 0 < len(argv) {
1777         if argv[0] == "-s" {
1778             debug = false
1779             argv = argv[1:]
1780         }
1781     }
1782     if 0 < len(argv) {
1783         argv = argv[1:]
1784     }
1785     port, err := net.ResolveTCPAddr("tcp",local);
1786     if err != nil {
1787         fmt.Printf("--En- S: Address error: %s (%s)\n",local,err)
1788         return
1789     }
1790     fmt.Printf(Elapsed(Start)+"--In- S: Listening at %s...\n",local);
1791     sconn, err := net.ListenTCP("tcp", port)
1792     if err != nil {
1793         fmt.Printf(Elapsed(Start)+"--En- S: Listen error: %s (%s)\n",local,err)
1794         return
1795     }
1796
1797     reqbuf := make([]byte,LINESIZE)
1798     res := ""
1799     for {
1800         fmt.Printf(Elapsed(Start0)+"--In- S: Listening at %s...\n",local);
1801         aconn, err := sconn.AcceptTCP()
1802         Start = time.Now()
1803         if err != nil {
1804             fmt.Printf(Elapsed(Start)+"--En- S: Accept error: %s (%s)\n",local,err)
1805             return
1806         }
1807         clnt, _ := aconn.File()
1808         fd := clnt.Fd()
1809         ar := aconn.RemoteAddr()
1810         if debug { fmt.Printf(Elapsed(Start0)+"--In- S: Accepted TCP at %s [%d] <- %v\n",
1811                     local,fd,ar) }
1812         res = fmt.Sprintf("220 GShell/%s Server\r\n",VERSION)
1813         fmt.Fprintf(clnt,"%s",res)
1814         if debug { fmt.Printf(Elapsed(Start)+"--In- S: %s",res) }
1815         count, err := clnt.Read(reqbuf)
1816         if err != nil {
1817             fmt.Printf(Elapsed(Start)+"--En- C: (%v %v) %v",
1818                         count,err,string(reqbuf))
1819         }
1820         req := string(reqbuf[:count])
1821         if debug { fmt.Printf(Elapsed(Start)+"--In- C: %v",string(req)) }
1822         req := strings.Split(string(req),"\\r")
1823         cmdv := gshScanArg(req[0],0)
1824         //cmdv := strings.Split(reqv[0]," ")
1825         switch cmdv[0] {
1826             case "HELO":
1827                 res = fmt.Sprintf("250 %v",req)
1828             case "GET":
1829                 // download {remotefile|-zN} [localfile]
1830                 var dszie int64 = 32*1024*1024
1831                 var bsize int = 64*1024
1832                 var fname string = ""
1833                 var in *os.File = nil
1834                 var pseudoEOF = false
1835                 if 1 < len(cmdv) {
1836                     fname = cmdv[1]
1837                     if strBegins(fname,"-z") {
1838                         fmt.Sscanf(fname[2:], "%d", &dszie)
1839                     }else
1840                         if strBegins(fname,"{") {
1841                             xin,xout,err := gsh.Popen(fname,"r")
1842                             if err {
1843                             }else{
1844                                 xout.Close()
1845                                 defer xin.Close()
1846                                 in = xin
1847                                 dszie = MaxStreamSize
1848                                 pseudoEOF = true
1849                             }
1850                         }else{
1851                             xin,err := os.Open(fname)
1852                             if err != nil {
1853                                 fmt.Printf("--En- GET (%v)\n",err)
1854                             }else{
1855                                 defer xin.Close()
1856                                 in = xin
1857                                 fi,_ := xin.Stat()
1858                                 dszie = fi.Size()
1859                             }
1860                         }
1861                     //fmt.Printf(Elapsed(Start)+"--In- GET %v:%v\n",dszie,bsize)
1862                     res = fmt.Sprintf("200 %v\r\n",dszie)
1863                     fmt.Fprintf(clnt,"%v",res)
1864                     tcpPush(clnt); // should be separated as line in receiver
1865                     fmt.Printf(Elapsed(Start)+"--In- S: %v",res)
1866                     wcount := fileRelay("SendGET",in,clnt,dszie,bsize)
1867                     if pseudoEOF {
1868                         in.Close() // pipe from the command
1869                         // show end of stream data (its size) by OOB?
1870                         SoftEOF := fmt.Sprintf("(SoftEOF %v)",wcount)
1871                         fmt.Printf(Elapsed(Start)+"--In- S: Send %v\n",SoftEOF)
1872                     }
1873                     tcpPush(clnt); // to let SoftEOF data apper at the top of recevied data
1874

```

```

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

```

```

2000     xin,xout,err := gsh.Popen(fname,"w")
2001     if err {
2002     }else{
2003         xin.Close()
2004         defer xout.Close()
2005         out = xout
2006         out_tobeclosed = xout
2007         pid = 0 // should be its pid
2008     }
2009 }else{
2010     // should write to temporary file
2011     // should suppress ^C on tty
2012     xout,err := os.OpenFile(fname,os.O_CREATE|os.O_RDWR|os.O_TRUNC,0600)
2013     if err != nil {
2014         fmt.Println("--En- %v\n",err)
2015     }
2016     out = xout
2017     //fmt.Printf("--In-- %d > %s\n",out.Fd(),fname)
2018 }
2019 in,_ := serv.File()
2020 fileRelay("RecvGET",in,out,dsiz,bsize)
2021 if 0 <= pid {
2022     gsh.gshPA = savPA // recovery of Fd(), and more?
2023     fmt.Printf(Elapsed(Start)+"--In- L: close Pipe > %v\n",fname)
2024     out_tobeclosed.Close()
2025     //syscall.Wait4(pid,nil,0,nil) //@@
2026 }
2027 }
2028 }
2029 }else
2030 if argv[0] == "PUT" {
2031     remote,_ := serv.File()
2032     var local *os.File = nil
2033     var dsiz int64 = 32*1024*1024
2034     var bsize int = 64*1024
2035     var ofile string = "-"
2036     //fmt.Printf("--I-- Rex %v\n",argv)
2037     if 1 < len(argv) {
2038         fname := argv[1]
2039         if strBegins(fname,"-z") {
2040             fmt.Sscanf(fname[2:], "%d",&dsiz)
2041         }else
2042         if strBegins(fname,"(") {
2043             xin,xout,err := gsh.Popen(fname,"r")
2044             if err {
2045                 }else{
2046                     xout.Close()
2047                     defer xin.Close()
2048                     //in = xin
2049                     local = xin
2050                     fmt.Printf("--In- %d < Upload output of %v\n",
2051                         local.Fd(),fname)
2052                     ofile = "-from."+fname
2053                     dsiz = MaxStreamSize
2054                 }
2055             }else{
2056                 xlocal,err := os.Open(fname)
2057                 if err != nil {
2058                     fmt.Println("--En- (%s)\n",err)
2059                     local = nil
2060                 }else{
2061                     local = xlocal
2062                     fi,_ := local.Stat()
2063                     dsiz = fi.Size()
2064                     defer local.Close()
2065                     //fmt.Printf("--I-- Rex in(%v / %v)\n",ofile,dsiz)
2066                 }
2067                 ofile = fname
2068                 fmt.Printf(Elapsed(Start)+"--In- L: open(%v,r)=%v %v (%v)\n",
2069                         fname,dsiz,local,err)
2070             }
2071     }if 2 < len(argv) && argv[2] != "" {
2072         ofile = argv[2]
2073         //fmt.Printf("(%d)%v B.ofile=%v\n",len(argv),argv,ofile)
2074     }
2075     //fmt.Printf(Elapsed(Start)+"--I-- Rex out(%v)\n",ofile)
2076     fmt.Printf(Elapsed(Start)+"--In- PUT %v (%v)\n",dsiz,bsize)
2077     req = fmt.Sprintf("PUT %v %v (%v)",dsiz,ofile)
2078     if debug { fmt.Printf(Elapsed(Start)+"--In- C: %v",req) }
2079     fmt.Fprintf(serv,"%v",req)
2080     count,err = serv.Read(res)
2081     if debug { fmt.Printf(Elapsed(Start)+"--In- S: %v",string(res[0:count])) }
2082     fileRelay("SendPUT",local,remote,dsiz,bsize)
2083 }else{
2084     req = fmt.Sprintf("%v\r\n",strings.Join(argv, " "))
2085     if debug { fmt.Printf(Elapsed(Start)+"--In- C: %v",req) }
2086     fmt.Fprintf(serv,"%v",req)
2087     //fmt.Printf("--In- sending RexRequest(%v)\n",len(req))
2088 }
2089 //fmt.Printf(Elapsed(Start)+"--In- waiting RexResponse...\n")
2090 count,err = serv.Read(res)
2091 ress := ""
2092 if count == 0 {
2093     ress = "(nil)\r\n"
2094 }else{
2095     ress = string(res[:count])
2096 }
2097 if err != nil {
2098     fmt.Printf(Elapsed(Start)+"--En- S: (%d,%v) %v",count,err,ress)
2099 }else{
2100     fmt.Printf(Elapsed(Start)+"--In- S: %v",ress)
2101 }
2102 serv.Close()
2103 //conn.Close()
2104
2105 var stat string
2106 var rcode int
2107 fmt.Sscanf(ress,"%d %s",&rcode,&stat)
2108 //fmt.Printf("--D-- Client: %v (%v)",rcode,stat)
2109 return rcode,ress
2110
2111 }
2112
2113 // <a name="remote-sh">Remote Shell</a>
2114 // gcp file [...] { [host]:[port]:[dir] | dir } // -p | -no-p
2115 func (gsh*GshContext)FileCopy(argv[]string){
2116     var host = ""
2117     var port = ""
2118     var upload = false
2119     var download = false
2120     var xargv = []string{"rex-gcp"}
2121     var srcv = []string{}
2122     var dstv = []string{}
2123     argv = argv[1:]

```

```

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

```

```

2250
2251  /*
2252  nargv := gshScanArg(strings.Join(argv, " "),0)
2253  fmt.Printf("--D-- nargc=%d [%v]\n",len(nargv),nargv)
2254  if nargv[1][0] == '{' {
2255      nargv[1] = "(" + nargv[1] + ")"
2256      fmt.Printf("--D-- nargc=%d [%v]\n",len(nargv),nargv)
2257  }
2258  argv = nargv
2259  */
2260  nargv := []string{}
2261  nargv = append(nargv,""+strings.Join(argv[1:], " ")+"")
2262  fmt.Printf("--D-- nargc=%d %v\n",len(nargv),nargv)
2263  argv = nargv
2264
2265  xargv := []string{"rex-exec","@"+gsh.RSERV,"GET"}
2266  xargv = append(xargv,argv...)
2267  xargv = append(xargv,"/dev/tty")
2268  rcode,stat := gsh.RexecClient(xargv)
2269  if (rcode / 100) == 2 {
2270      fmt.Printf("--I-- OK Rexec (%v) %v\n",rcode,stat)
2271  }else{
2272      fmt.Printf("--I-- NG Rexec (%v) %v\n",rcode,stat)
2273  }
2274 }
2275 func (gsh*GshContext)Rchdir(argv[]string){
2276  if len(argv) <= 1 {
2277      return
2278  }
2279  cwd, _ := os.Getwd()
2280  os.Chdir(gsh.RWD)
2281  os.Chdir(argv[1])
2282  twd,_ := os.Getwd()
2283  gsh.RWD = twd
2284  fmt.Printf("--I-- JWD=%v\n",twd)
2285  os.Chdir(cwd)
2286 }
2287 func (gsh*GshContext)Rpwd(argv[]string){
2288  fmt.Printf("%v",gsh.RWD)
2289 }
2290 func (gsh*GshContext)Rls(argv[]string){
2291  cwd, _ := os.Getwd()
2292  os.Chdir(gsh.RWD)
2293  argv[0] = "-ls"
2294  gsh.xfind(argv)
2295  os.Chdir(cwd)
2296 }
2297 func (gsh*GshContext)Rput(argv[]string){
2298  var local string = ""
2299  var remote string = ""
2300  if 1 < len(argv) {
2301      local = argv[1]
2302      remote = local // base name
2303  }
2304  if 2 < len(argv) {
2305      remote = argv[2]
2306  }
2307  fmt.Printf("--I-- jput from=%v to=%v\n",local,gsh.Trepath(remote))
2308 }
2309 func (gsh*GshContext)Rget(argv[]string){
2310  var remote string = ""
2311  var local string = ""
2312  if 1 < len(argv) {
2313      remote = argv[1]
2314      local = remote // base name
2315  }
2316  if 2 < len(argv) {
2317      local = argv[2]
2318  }
2319  fmt.Printf("--I-- jget from=%v to=%v\n",gsh.Trepath(remote),local)
2320 }
2321
2322 // <a name="network">network</a>
2323 // -s, -sl, -so // bi-directional, source, sync (maybe socket)
2324 func (gshCtxx*GshContext)sconnect(inTCP bool, argv []string) {
2325  gshPA := gshCtxx.gshPA
2326  if len(argv) < 2 {
2327      fmt.Printf("Usage: -s [host]:[port[.udp]]\n")
2328      return
2329  }
2330  remote := argv[1]
2331  if remote == ":" { remote = "0.0.0.0:9999" }
2332
2333  if inTCP { // TCP
2334      dport, err := net.ResolveTCPAddr("tcp",remote);
2335      if err != nil {
2336          fmt.Printf("Address error: %s (%s)\n",remote,err)
2337          return
2338      }
2339      conn, err := net.DialTCP("tcp",nil,dport)
2340      if err != nil {
2341          fmt.Printf("Connection error: %s (%s)\n",remote,err)
2342          return
2343      }
2344      file, _ := conn.File();
2345      fd := file.Fd()
2346      fmt.Printf("Socket: connected to %s, socket[%d]\n",remote,fd)
2347
2348      savfd := gshPA.Files[1]
2349      gshPA.Files[1] = fd;
2350      gshCtxx.gshellv(argv[2:])
2351      gshPA.Files[1] = savfd
2352      file.Close()
2353      conn.Close()
2354  }else{
2355      //dport, err := net.ResolveUDPPort("udp4",remote);
2356      dport, err := net.ResolveUDPAddr("udp",remote);
2357      if err != nil {
2358          fmt.Printf("Address error: %s (%s)\n",remote,err)
2359          return
2360      }
2361      //conn, err := net.DialUDP("udp4",nil,dport)
2362      conn, err := net.DialUDP("udp",nil,dport)
2363      if err != nil {
2364          fmt.Printf("Connection error: %s (%s)\n",remote,err)
2365          return
2366      }
2367      file, _ := conn.File();
2368      fd := file.Fd()
2369
2370      ar := conn.RemoteAddr()
2371      //al := conn.LocalAddr()
2372      fmt.Printf("Socket: connected to %s [%s], socket[%d]\n",
2373                  remote,ar.String(),fd)
2374 }
```

```

2375     savfd := gshPA.Files[1]
2376     gshPA.Files[1] = fd;
2377     gshCtx.gshellv(argv[2:])
2378     gshPA.Files[1] = savfd
2379     file.Close()
2380     conn.Close()
2381   }
2382 }
2383 func (gshCtx*GshContext)xaccept(inTCP bool, argv []string) {
2384   gshPA := gshCtx.gshPA
2385   if len(argv) < 2 {
2386     fmt.Printf("Usage: -ac [host]:[port[.udp]]\n")
2387     return
2388   }
2389   local := argv[1]
2390   if local == ":" { local = "0.0.0.0:9999" }
2391   if inTCP { // TCP
2392     port, err := net.ResolveTCPAddr("tcp",local);
2393     if err != nil {
2394       fmt.Printf("Address error: %s (%s)\n",local,err)
2395       return
2396     }
2397     //fmt.Printf("Listen at %s...\n",local);
2398     sconn, err := net.ListenTCP("tcp", port)
2399     if err != nil {
2400       fmt.Printf("Listen error: %s (%s)\n",local,err)
2401       return
2402     }
2403     //fmt.Printf("Accepting at %s...\n",local);
2404     aconn, err := sconn.AcceptTCP()
2405     if err != nil {
2406       fmt.Printf("Accept error: %s (%s)\n",local,err)
2407       return
2408     }
2409     file, _ := aconn.File()
2410     fd := file.Fd()
2411     fmt.Printf("Accepted TCP at %s [%d]\n",local,fd)
2412
2413     savfd := gshPA.Files[0]
2414     gshPA.Files[0] = fd;
2415     gshCtx.gshellv(argv[2:])
2416     gshPA.Files[0] = savfd
2417
2418     sconn.Close();
2419     aconn.Close();
2420     file.Close();
2421 }else{
2422   //port, err := net.ResolveUDPAddr("udp4",local);
2423   port, err := net.ResolveUDPAddr("udp",local);
2424   if err != nil {
2425     fmt.Printf("Address error: %s (%s)\n",local,err)
2426     return
2427   }
2428   fmt.Printf("Listen UDP at %s...\n",local);
2429   //uconn, err := net.ListenUDP("udp4", port)
2430   uconn, err := net.ListenUDP("udp", port)
2431   if err != nil {
2432     fmt.Printf("Listen error: %s (%s)\n",local,err)
2433     return
2434   }
2435   file, _ := uconn.File()
2436   fd := file.Fd()
2437   ar := uconn.RemoteAddr()
2438   remote := ""
2439   if ar != nil { remote = ar.String() }
2440   if remote == "" { remote = "?" }
2441
2442   // not yet received
2443   //fmt.Printf("Accepted at %s [%d] <- %s\n",local,fd,"")
2444
2445   savfd := gshPA.Files[0]
2446   gshPA.Files[0] = fd;
2447   savenv := gshPA.Env
2448   gshPA.Env = append(savenv, "REMOTE_HOST="+remote)
2449   gshCtx.gshellv(argv[2:])
2450   gshPA.Env = savenv
2451   gshPA.Files[0] = savfd
2452
2453   uconn.Close();
2454   file.Close();
2455 }
2456 }
2457 // empty line command
2458 func (gshCtx*GshContext)xPwd(argv[]string){
2459   // execute context command, pwd + date
2460   // context notation, representation scheme, to be resumed at re-login
2461   // cwd, _ := os.Getwd()
2462   switch {
2463   case isin("-a",argv):
2464     gshCtx.ShowChdirHistory(argv)
2465   case isin("-ls",argv):
2466     showFileInfo(cwd,argv)
2467   default:
2468     fmt.Printf("%s\n", cwd)
2469   case isin("-v",argv): // obsolete emtpy command
2470     t := time.Now()
2471     date := t.Format(time.UnixDate)
2472     exe, _ := os.Executable()
2473     host, _ := os.Hostname()
2474     fmt.Printf("PWD=%s\n", cwd)
2475     fmt.Printf("HOST=%s\n", host)
2476     fmt.Printf("DATE=%s\n", date)
2477     fmt.Printf("TIME=%s\n", t.String())
2478     fmt.Printf("PID=%d\n", os.Getpid())
2479     fmt.Printf("EXE=%s\n", exe)
2480     fmt.Printf("{}\n")
2481   }
2482 }
2483 }
2484
2485 // <a name="history">History</a>
2486 // these should be browsed and edited by HTTP browser
2487 // show the time of command with -t and direcotry with -ls
2488 // openfile-history, sort by -a -m -c
2489 // sort by elapsed time by -t -s
2490 // search by "more" like interface
2491 // edit history
2492 // sort history, and wc or uniq
2493 // CPU and other resource consumptions
2494 // limit showing range (by time or so)
2495 // export / import history
2496 func (gshCtx *GshContext)xHistory(argv []string){
2497   atWorkDirX := -1
2498   if 1 < len(argv) && strBegins(argv[1],"@") {
2499     atWorkDirX,_ = strconv.Atoi(argv[1][1:])

```

```

2500 }
2501 //fmt.Printf("--D-- showHistory(%v)\n",argv)
2502 for i, v := range gshCtx.CommandHistory {
2503     // exclude commands not to be listed by default
2504     // internal commands may be suppressed by default
2505     if v.CmdLine == "" && !isin("-a",argv) {
2506         continue;
2507     }
2508     if 0 <= atWorkDirX {
2509         if v.WorkDirX != atWorkDirX {
2510             continue
2511         }
2512     }
2513     if !isin("-n",argv){ // like "fc"
2514         fmt.Printf("!%s-2d ",i)
2515     }
2516     if isin("-v",argv){
2517         fmt.Println(v) // should be with it date
2518     }else{
2519         if isin("-l",argv) || isin("-10",argv) {
2520             elps := v.EndAt.Sub(v.StartAt);
2521             start := v.StartAt.Format(time.Stamp)
2522             fmt.Printf("%d ",v.WorkDirX)
2523             fmt.Printf("[%v] %11v/t ",start,elps)
2524         }
2525         if isin("-1",argv) && !isin("-10",argv){
2526             fmt.Printf("%v",Rusagef("%t %u\t// %s",argv,v.Rusage))
2527         }
2528         if isin("-at",argv) { // isin("-ls",argv)
2529             dhi := v.WorkDirX // workdir history index
2530             fmt.Printf("%d %s\t",dhi,v.Workdir)
2531             // show the FileInfo of the output command??
2532         }
2533         fmt.Printf("%s",v.CmdLine)
2534         fmt.Printf("\n")
2535     }
2536 }
2537 }
2538 // !n - history index
2539 func searchHistory(gshCtx GshContext, gline string) (string, bool, bool){
2540     if gline[0] == '!' {
2541         hix, err := strconv.Atoi(gline[1:])
2542         if err != nil {
2543             fmt.Printf("--E-- (%s : range)\n",hix)
2544             return "", false, true
2545         }
2546         if hix < 0 || len(gshCtx.CommandHistory) <= hix {
2547             fmt.Printf("--E-- (%d : out of range)\n",hix)
2548             return "", false, true
2549         }
2550         return gshCtx.CommandHistory[hix].CmdLine, false, false
2551     }
2552     // search
2553     //for i, v := range gshCtx.CommandHistory {
2554     //}
2555     return gline, false, false
2556 }
2557 func (gsh*GshContext)cmdStringInHistory(hix int)(cmd string, ok bool){
2558     if 0 <= hix && hix < len(gsh.CommandHistory) {
2559         return gsh.CommandHistory[hix].CmdLine,true
2560     }
2561     return "",false
2562 }
2563
2564 // temporary adding to PATH environment
2565 // cd name -lib for LD_LIBRARY_PATH
2566 // chdir with directory history (date + full-path)
2567 // -s for sort option (by visit date or so)
2568 func (gsh*GshContext>ShowChdirHistory(i int,v GChdirHistory, argv []string){
2569     fmt.Printf("%s-2d ",v.CmdIndex) // the first command at this WorkDir
2570     fmt.Printf("%d ",i)
2571     fmt.Printf("[%v] ",v.MovedAt.Format(time.Stamp))
2572     showFileInfo(v.dir,argv)
2573 }
2574 func (gsh*GshContext>ShowChdirHistory(argv []string){
2575     for i, v := range gsh.CkdirHistory {
2576         gsh.ShowChdirHistory1(i,v,argv)
2577     }
2578 }
2579 func skipOpts(argv[]string)(int){
2580     for i,v := range argv {
2581         if strBegins(v,"-") {
2582             }else{
2583                 return i
2584             }
2585     }
2586     return -1
2587 }
2588 func (gshCtx*GshContext)xChdir(argv []string){
2589     cdhist := gshCtx.CkdirHistory
2590     if isin("?",argv) || isin("-t",argv) || isin("-a",argv) {
2591         gshCtx.ShowChdirHistory(argv)
2592         return
2593     }
2594     pwd, _ := os.Getwd()
2595     dir := ""
2596     if len(argv) <= 1 {
2597         dir = toFullpath("~")
2598     }else{
2599         i := skipOpts(argv[1:])
2600         if i < 0 {
2601             dir = toFullpath("~")
2602         }else{
2603             dir = argv[1+i]
2604         }
2605     }
2606     if strBegins(dir,"@") {
2607         if dir == "@0" { // obsolete
2608             dir = gshCtx.StartDir
2609         }else{
2610             if dir == "@!" {
2611                 index := len(cdhist) - 1
2612                 if 0 < index { index -= 1 }
2613                 dir = cdhist[index].dir
2614             }else{
2615                 index, err := strconv.Atoi(dir[1:])
2616                 if err != nil {
2617                     fmt.Printf("--E-- xChdir(%v)\n",err)
2618                     dir = "?"
2619                 }else{
2620                     if len(gshCtx.CkdirHistory) <= index {
2621                         fmt.Printf("--E-- xChdir(history range error)\n")
2622                         dir = "?"
2623                     }else{
2624                         dir = cdhist[index].Dir
2625                     }
2626                 }
2627             }
2628         }
2629     }
2630 }
```

```

2625     }
2626 }
2627 if dir != "?" {
2628     err := os.Chdir(dir)
2629     if err != nil {
2630         fmt.Printf("--E-- xChdir(%s)(%v)\n", argv[1], err)
2631     }else{
2632         cwd, _ := os.Getwd()
2633         if cwd != pwd {
2634             hist1 := GCChdirHistory { }
2635             hist1.Dir = cwd
2636             hist1.Movedat = time.Now()
2637             hist1.CmdIndex = len(gshCtx.CommandHistory)+1
2638             gshCtx.ChdirHistory = append(cdhist,hist1)
2639             if !isin("-s",argv){
2640                 //cwd, _ := os.Getwd()
2641                 //fmt.Printf("%s\n", cwd)
2642                 ix := len(gshCtx.ChdirHistory)-1
2643                 gshCtx.ShowChdirHistoryl(ix,hist1,argv)
2644             }
2645         }
2646     }
2647 }
2648 if isin("-ls",argv){
2649     cwd, _ := os.Getwd()
2650     showFileInfo(cwd,argv);
2651 }
2652 }
2653 func TimeValSub(tv1 *syscall.Timeval, tv2 *syscall.Timeval){
2654     *tv1 = syscall.NsecToTimeval(tv1.Nano() - tv2.Nano())
2655 }
2656 func RusageSub(rul1, ru2 [2]syscall.Rusage)([2]syscall.Rusage){
2657     TimeValSub(&rul1[0].Utime,&ru2[0].Utime)
2658     TimeValSub(&rul1[0].Stime,&ru2[0].Stime)
2659     TimeValSub(&rul1[1].Utime,&ru2[1].Utime)
2660     TimeValSub(&rul1[1].Stime,&ru2[1].Stime)
2661     return rul1
2662 }
2663 func TimeValAdd(tv1 syscall.Timeval, tv2 syscall.Timeval)(syscall.Timeval){
2664     tvs := syscall.NsecToTimeval(tv1.Nano() + tv2.Nano())
2665     return tvs
2666 }
2667 */
2668 func RusageAddv(rul1, ru2 [2]syscall.Rusage)([2]syscall.Rusage){
2669     TimeValAdd(rul1[0].Utime,ru2[0].Utime)
2670     TimeValAdd(rul1[0].Stime,ru2[0].Stime)
2671     TimeValAdd(rul1[1].Utime,ru2[1].Utime)
2672     TimeValAdd(rul1[1].Stime,ru2[1].Stime)
2673     return rul1
2674 }
2675 */
2676 */
2677 // <a name="rusage">Resource Usage</a>
2678 func SRusagef(fmtspec string, argv []string, ru [2]syscall.Rusage)(string){
2679     // ru[0] self , ru[1] children
2680     ut := TimeValAdd(ru[0].Utime,ru[1].Utime)
2681     st := TimeValAdd(ru[0].Stime,ru[1].Stime)
2682     uu := (ut.Sec*1000000 + int64(ut.Usec)) * 1000
2683     su := (st.Sec*1000000 + int64(st.Usec)) * 1000
2684     tu := uu + su
2685     ret := fmt.Sprintf(" %v/sum",abbttime(tu))
2686     ret += fmt.Sprintf(" %v/usr",abbttime(uu))
2687     ret += fmt.Sprintf(" %v/sys",abbttime(su))
2688     return ret
2689 }
2690 func Rusagef(fmtspec string, argv []string, ru [2]syscall.Rusage)(string){
2691     ut := TimeValAdd(ru[0].Utime,ru[1].Utime)
2692     st := TimeValAdd(ru[0].Stime,ru[1].Stime)
2693     fmt.Printf("%d.%06ds/u ",ut.Sec,ut.Usec) //ru[1].Utime.Sec,ru[1].Utime.Usec)
2694     fmt.Printf("%d.%06ds/s ",st.Sec,st.Usec) //ru[1].Stime.Sec,ru[1].Stime.Usec)
2695     return ""
2696 }
2697 func Getrusagev(([2]syscall.Rusage)){
2698     var rvv = [2]syscall.Rusage{}
2699     syscall.Getrusage(syscall.RUSAGE_SELF,&rvv[0])
2700     syscall.Getrusage(syscall.RUSAGE_CHILDREN,&rvv[1])
2701     return rvv
2702 }
2703 */
2704 func ShowRusage(what string,argv []string, ru *syscall.Rusage){
2705     fmt.Printf(" %s: ",what);
2706     fmt.Printf(" %s=%d.%06ds",ru.Utime.Sec,ru.Utime.Usec)
2707     fmt.Printf(" %s=%d.%06ds",ru.Stime.Sec,ru.Stime.Usec)
2708     fmt.Printf(" %s=%vB",ru.Maxrss)
2709     if isin("-l",argv) {
2710         fmt.Printf(" MinFlt=%v",ru.Minflt)
2711         fmt.Printf(" MajFlt=%v",ru.Majflt)
2712         fmt.Printf(" IXRSS=%vB",ru.Ixrss)
2713         fmt.Printf(" IDRSS=%vB",ru.Idrss)
2714         fmt.Printf(" Nswap=%vB",ru.Nswap)
2715         fmt.Printf(" Read=%v",ru.Inblock)
2716         fmt.Printf(" Write=%v",ru.Outblock)
2717     }
2718     fmt.Printf(" %s=%v",ru.Msgsnd)
2719     fmt.Printf(" %s=%v",ru.Msgrcv)
2720     //if isin("-l",argv) {
2721     fmt.Printf(" %s=%v",ru.Nsignals)
2722     //}
2723     fmt.Printf("\n");
2724 }
2725 func (gshCtx *GshContext)xTime(argv[]string)(bool){
2726     if 2 <= len(argv){
2727         gshCtx.LastRusage = syscall.Rusage{}
2728         rusagev1 := Getrusagev()
2729         fin := gshCtx.gshellv(argv[1:])
2730         rusagev2 := Getrusagev()
2731         showRusage(argv[1],argv,&gshCtx.LastRusage)
2732         rusagev1 := RusageSubv(rusagev2,rusagev1)
2733         showRusage("self",argv,&rusagev1)
2734         showRusage("chld",argv,&rusagev1)
2735         return fin
2736     }else{
2737         rusage:= syscall.Rusage {}
2738         syscall.Getrusage(syscall.RUSAGE_SELF,&rusage)
2739         showRusage("self",argv,&rusage)
2740         syscall.Getrusage(syscall.RUSAGE_CHILDREN,&rusage)
2741         showRusage("chld",argv,&rusage)
2742         return false
2743     }
2744 }
2745 func (gshCtx *GshContext)xJobs(argv[]string){
2746     fmt.Printf("%d Jobs\n",len(gshCtx.BackGroundJobs))
2747     for j1, pid := range gshCtx.BackGroundJobs {
2748         //wstat := syscall.WaitStatus {0}
2749         rusage := syscall.Rusage {}

```

```

2750 //wpid, err := syscall.Wait4(pid,&wstat,syscall.WNOHANG,&rusage);
2751 wpid, err := syscall.Wait4(pid,nil,syscall.WNOHANG,&rusage);
2752 if err != nil {
2753     fmt.Printf("--E-- %d [%d] (%v)\n",ji.pid,err)
2754 }else{
2755     fmt.Printf("%d[%d]\n",ji.pid,wpid)
2756     showRusage("child",argv,&rusage)
2757 }
2758 }
2759 }
2760 func (gsh*GshContext)inBackground(argv[]string)(bool){
2761     if gsh.CmdTrace { fmt.Printf("--I-- inBackground(%v)\n",argv) }
2762     gsh.BackGround = true // set background option
2763     xfin := false
2764     xfin = gsh.gshellv(argv)
2765     gsh.BackGround = false
2766     return xfin
2767 }
2768 // -o file without command means just opening it and refer by #N
2769 // should be listed by "files" command
2770 func (gshCtx*GshContext)xOpen(argv[]string){
2771     var pv = []int{-1,-1}
2772     err := syscall.Pipe(pv)
2773     fmt.Printf("--I-- pipe()=[#%d,#%d](%v)\n",pv[0],pv[1],err)
2774 }
2775 func (gshctx*GshContext)fromPipe(argv[]string){
2776 }
2777 func (gshctx*GshContext)xClose(argv[]string){
2778 }
2779
2780 // <a name="redirect">redirect</a>
2781 func (gshCtx*GshContext)redirect(argv[]string)(bool){
2782     if len(argv) < 2 {
2783         return false
2784     }
2785     cmd := argv[0]
2786     fname := argv[1]
2787     var file *os.File = nil
2788
2789     ffix := 0
2790     mode := os.O_RDONLY
2791
2792     switch {
2793     case cmd == "-i" || cmd == "<":
2794         ffix = 0
2795         mode = os.O_RDONLY
2796     case cmd == "-o" || cmd == ">":
2797         ffix = 1
2798         mode = os.O_RDWR | os.O_CREATE
2799     case cmd == "-a" || cmd == ">>":
2800         ffix = 1
2801         mode = os.O_RDWR | os.O_CREATE | os.O_APPEND
2802     }
2803     if fname[0] == '#' {
2804         fd, err := strconv.Atoi(fname[1:])
2805         if err != nil {
2806             fmt.Printf("--E-- (%v)\n",err)
2807             return false
2808         }
2809         file = os.NewFile(uintptr(fd),"MaybePipe")
2810     }else{
2811         xfile, err := os.OpenFile(argv[1], mode, 0600)
2812         if err != nil {
2813             fmt.Printf("--E-- (%s)\n",err)
2814             return false
2815         }
2816         file = xfile
2817     }
2818     gshPA := gshCtx.gshPA
2819     savfd := gshPA.Files[ffix]
2820     gshPA.Files[ffix] = file.Fd()
2821     fmt.Printf("--I-- Opened [%d] %s\n",file.Fd(),argv[1])
2822     gshctx.gshell(argv[2:])
2823     gshPA.Files[ffix] = savfd
2824
2825     return false
2826 }
2827 }
2828
2829 //fmt.Fprintf(res, "GShell Status: %q", html.EscapeString(req.URL.Path))
2830 func httpHandler(res http.ResponseWriter, req *http.Request){
2831     path := req.URL.Path
2832     fmt.Printf("--I-- Got HTTP Request(%s)\n",path)
2833     {
2834         gshCtxBuf, _ := setupGshContext()
2835         gshCtx := &gshCtxBuf
2836         fmt.Printf("-i-- %s\n",path[1:])
2837         gshCtx.tgshell(path[1:])
2838     }
2839     fmt.Fprintf(res, "Hello(^~^)/\n%s\n",path)
2840 }
2841 func (gshCtx *GshContext) httpServer(argv []string){
2842     http.HandleFunc("/", httpHandler)
2843     accport := "localhost:9999"
2844     fmt.Printf("--I-- HTTP Server Start at [%s]\n",accport)
2845     http.ListenAndServe(accport,nil)
2846 }
2847 func (gshCtx *GshContext)xGo(argv[]string){
2848     go gshCtx.gshellv(argv[1:]);
2849 }
2850 func (gshCtx *GshContext) xPs(argv[]string)(){
2851 }
2852
2853 // <a name="plugin">Plugin</a>
2854 // plugin [-ls [names]] to list plugins
2855 // Reference: <a href="https://golang.org/src/plugin/">plugin</a> source code
2856 func (gshCtx *GshContext) whichPlugin(name string,argv[]string)(pi *PluginInfo){
2857     pi = nil
2858     for _,p := range gshCtx.PluginFuncs {
2859         if p.Name == name && pi == nil {
2860             pi = &p
2861         }
2862         if lisin("-a",argv){
2863             //fmt.Printf("%v %v ",i,p)
2864             if lisin("-ls",argv){
2865                 showFileInfo(p.Path,argv)
2866             }else{
2867                 fmt.Printf("%s\n",p.Name)
2868             }
2869         }
2870     }
2871     return pi
2872 }
2873 func (gshCtx *GshContext) xPlugin(argv[]string) (error) {
2874     if len(argv) == 0 || argv[0] == "-ls" {

```

```

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

```

```

3000         gsh.printVal(curfmt,gsh.iValues[hi],optv)
3001         outlen += 1 // should be the real length
3002     }else{
3003         fmt.Printf("(out-range)")
3004     }
3005     xi += leng
3006     continue;
3007   }
3008 }
3009 fmt.Printf("%c",fch)
3010 outlen += 1
3111 }
3112 }else{
3113   //fmt.Printf("--D- print %s\n")
3114   for i,v := range list {
3115     if 0 < i {
3116       fmt.Printf(div)
3117     }
3118     gsh.printVal(curfmt,v,optv)
3119     outlen += 1
3120   }
3121 }
3122 if 0 < outlen {
3123   fmt.Printf("\n")
3124 }
3125 }
3126 func (gsh*GshContext)Scanv(argv[]string){
3127   //fmt.Printf("--D-- Scanv(%v)\n",argv)
3128   if len(argv) == 1 {
3129     return
3130   }
3131   argv = argv[1:]
3132   fmts := ""
3133   if strBegins(argv[0],"-F") {
3134     fmts = argv[0]
3135     gsh.iDelimiter = fmts
3136     argv = argv[1:]
3137   }
3138   input := strings.Join(argv, " ")
3139   if fmts == "" { // simple decomposition
3140     v := scanv(input)
3141     gsh.iValues = v
3142     //fmt.Printf("%v\n",strings.Join(v, ""))
3143   }else{
3144     v := make([]string,8)
3145     n,err := fmt.Sscanf(input,fmts,&v[0],&v[1],&v[2],&v[3])
3146     fmt.Println("-D-- Scanf ->(%v) n=%d err=(%v)\n",v,n,err)
3147     gsh.iValues = v
3148   }
3149 }
3150 func (gsh*GshContext)Printv(argv[]string){
3151   if false { //@0U
3152     fmt.Printf("%v\n",strings.Join(argv[1:], " "))
3153   }
3154 }
3155 //fmt.Printf("--D-- Printv(%v)\n",argv)
3156 //fmt.Printf("%v\n",strings.Join(gsh.iValues,","))
3157 div := gsh.iDelimiter
3158 fmts := ""
3159 argv = argv[1:]
3160 if 0 < len(argv) {
3161   if strBegins(argv[0],"-F") {
3162     div = argv[0][2:]
3163     argv = argv[1:]
3164   }
3165 }
3166 optv := []string{}
3167 for _,v := range argv {
3168   if strBegins(v,"-"){
3169     optv = append(optv,v)
3170     argv = argv[1:]
3171   }else{
3172     break;
3173   }
3174 }
3175 if 0 < len(argv) {
3176   fmts = strings.Join(argv, " ")
3177 }
3178 gsh.printfv(fmts,div,argv,optv,gsh.iValues)
3179 }
3180 func (gsh*GshContext)Basename(argv[]string){
3181   for i,v := range gsh.iValues {
3182     gsh.iValues[i] = filepath.Base(v)
3183   }
3184 }
3185 }
3186 func (gsh*GshContext)Sortv(argv[]string){
3187   sv := gsh.iValues
3188   sort.Slice(sv , func(i,j int) bool {
3189     return sv[i] < sv[j]
3190   })
3191 }
3192 func (gsh*GshContext)Shiftv(argv[]string){
3193   vi := len(gsh.iValues)
3194   if 0 < vi {
3195     if isin("-r",argv) {
3196       top := gsh.iValues[0]
3197       gsh.iValues = append(gsh.iValues[1:],top)
3198     }else{
3199       gsh.iValues = gsh.iValues[1:]
3200     }
3201   }
3202 }
3203 }
3204 func (gsh*GshContext)Enq(argv[]string){
3205 }
3206 func (gsh*GshContext)Deq(argv[]string){
3207 }
3208 func (gsh*GshContext)Push(argv[]string){
3209   gsh.iValStack = append(gsh.iValStack,argv[1:])
3210   fmt.Printf("depth=%d\n",len(gsh.iValStack))
3211 }
3212 func (gsh*GshContext)Dump(argv[]string){
3213   for i,v := range gsh.iValStack {
3214     fmt.Printf("%d %v\n",i,v)
3215   }
3216 }
3217 func (gsh*GshContext)Pop(argv[]string){
3218   depth := len(gsh.iValStack)
3219   if 0 < depth {
3220     v := gsh.iValStack[depth-1]
3221     if isin("-cat",argv){
3222       gsh.iValues = append(gsh.iValues,v...)
3223     }else{
3224       gsh.iValues = v
3225     }
3226   }
3227 }
```

```

3125     }
3126     gsh.iValStack = gsh.iValStack[0:depth-1]
3127     fmt.Printf("depth=%d %s\n",len(gsh.iValStack),gsh.iValues)
3128 }else{
3129     fmt.Printf("depth=%d\n",depth)
3130 }
3131 }

3132 // <a name="interpreter">Command Interpreter</a>
3133 func (gshCtx*xGshContext)gshellv(argv []string) (fin bool) {
3134     fin = false
3135
3136     if gshCtx.CmdTrace { fmt.Fprintf(os.Stderr,"--I-- gshellv(%d)\n",len(argv)) }
3137     if len(argv) <= 0 {
3138         return false
3139     }
3140
3141     xargv := []string{}
3142     for ai := 0; ai < len(argv); ai++ {
3143         xargv = append(xargv,strsubst(gshCtx,argv[ai],false))
3144     }
3145     argv = xargv
3146     if false {
3147         for ai := 0; ai < len(argv); ai++ {
3148             fmt.Printf("[%d] %s [%d]\n",
3149                         ai,argv[ai],len(argv[ai]),argv[ai])
3150         }
3151 }
3152 cmd := argv[0]
3153 if gshCtx.CmdTrace { fmt.Fprintf(os.Stderr,"--I-- gshellv(%d)%v\n",len(argv),argv) }
3154 switch { // https://tour.golang.org/flowcontrol/11
3155 case cmd == "":
3156     gshCtx.xPwd([]string{}) // empty command
3157 case cmd == "-x":
3158     gshCtx.CmdTrace = ! gshCtx.CmdTrace
3159 case cmd == "-xt":
3160     gshCtx.CmdTime = ! gshCtx.CmdTime
3161 case cmd == "-ot":
3162     gshCtx.sconnect(true, argv)
3163 case cmd == "-on":
3164     gshCtx.sconnect(false, argv)
3165 case cmd == "-it":
3166     gshCtx.saccept(true , argv)
3167 case cmd == "-iu":
3168     gshCtx.saccept(false, argv)
3169 case cmd == "-i" || cmd == "<" || cmd == "-o" || cmd == ">" || cmd == "-a" || cmd == ">>" || cmd == "-s" || cmd == "><":
3170     gshCtx.redirect(argv)
3171 case cmd == "|":
3172     gshCtx.fromPipe(argv)
3173 case cmd == "args":
3174     gshCtx.Args(argv)
3175 case cmd == "bg" || cmd == "-bg":
3176     rfin := gshCtx.inBackground(argv[1:])
3177     return rfin
3178 case cmd == "-bn":
3179     gshCtx.Basename(argv)
3180 case cmd == "call":
3181     _ = gshCtx.excommand(false,argv[1:])
3182 case cmd == "cd" || cmd == "chdir":
3183     gshCtx.xChdir(argv);
3184 case cmd == "cksum":
3185     gshCtx.xFind(argv)
3186 case cmd == "-sum":
3187     gshCtx.xFind(argv)
3188 case cmd == "-sumtest":
3189     str := ""
3190     if 1 < len(argv) { str = argv[1] }
3191     crc := strCRC32(str,uint64(len(str)))
3192     fprintf(stderr,"%v\n",crc,len(str))
3193 case cmd == "close":
3194     gshCtx.xClose(argv)
3195 case cmd == "gcp":
3196     gshCtx.FileCopy(argv)
3197 case cmd == "dec" || cmd == "decode":
3198     gshCtx.Dec(argv)
3199 case cmd == "#define":
3200 case cmd == "dic" || cmd == "d":
3201     xDic(argv)
3202 case cmd == "dump":
3203     gshCtx.Dump(argv)
3204 case cmd == "echo" || cmd == "e":
3205     echo(argv,true)
3206 case cmd == "enc" || cmd == "encode":
3207     gshCtx.Enc(argv)
3208 case cmd == "env":
3209     env(argv)
3210 case cmd == "eval":
3211     xEval(argv[1:],true)
3212 case cmd == "ev" || cmd == "events":
3213     dumpEvents(argv)
3214 case cmd == "exec":
3215     _ = gshCtx.excommand(true,argv[1:])
3216     // should not return here
3217 case cmd == "exit" || cmd == "quit":
3218     // write Result code EXIT to 3>
3219     return true
3220 case cmd == "fdls":
3221     // dump the attributes of fds (of other process)
3222 case cmd == "find" || cmd == "fin" || cmd == "ufind" || cmd == "uf":
3223     gshCtx.xFind(argv[1:])
3224 case cmd == "fu":
3225     gshCtx.xFind(argv[1:])
3226 case cmd == "fork":
3227     // mainly for a server
3228 case cmd == "gen":
3229     gshCtx.gen(argv)
3230 case cmd == "-go":
3231     gshCtx.xGo(argv)
3232 case cmd == "grep":
3233     gshCtx.xFind(argv)
3234 case cmd == "gdeg":
3235     gshCtx.Deg(argv)
3236 case cmd == "geng":
3237     gshCtx.Eng(argv)
3238 case cmd == "gpop":
3239     gshCtx.Pop(argv)
3240 case cmd == "gpush":
3241     gshCtx.Push(argv)
3242 case cmd == "history" || cmd == "hi": // hi should be alias
3243     gshCtx.xHistory(argv)
3244 case cmd == "jobs":
3245     gshCtx.xJobs(argv)
3246 case cmd == "lisp" || cmd == "nlsp":
3247     gshCtx.SplitLine(argv)
3248 case cmd == "ls":
3249     gshCtx.xFind(argv)

```

```

3250     case cmd == "nop":
3251         // do nothing
3252     case cmd == "pipe":
3253         gshCtx.xOpen(argv)
3254     case cmd == "plug" || cmd == "plugin" || cmd == "pin":
3255         gshCtx.xPlugin(argv[1:])
3256     case cmd == "print" || cmd == "-pr":
3257         // output internal slice // also sprintf should be
3258         gshCtx.Println(argv)
3259     case cmd == "ps":
3260         gshCtx.xPs(argv)
3261     case cmd == "pstitle":
3262         // to be gsh.title
3263     case cmd == "rexec" || cmd == "rexd":
3264         gshCtx.RexecServer(argv)
3265     case cmd == "rexec" || cmd == "rex":
3266         gshCtx.RexecClient(argv)
3267     case cmd == "repeat" || cmd == "rep": // repeat cond command
3268         gshCtx.repeat(argv)
3269     case cmd == "replay":
3270         gshCtx.xReplay(argv)
3271     case cmd == "scan":
3272         // scan input (or so in fscanf) to internal slice (like Files or map)
3273         gshCtx.Scanv(argv)
3274     case cmd == "set":
3275         // set name ...
3276     case cmd == "serv":
3277         gshCtx.httpServer(argv)
3278     case cmd == "shift":
3279         gshCtx.Shiftv(argv)
3280     case cmd == "sleep":
3281         gshCtx.sleep(argv)
3282     case cmd == "-sort":
3283         gshCtx.Sortv(argv)
3284
3285     case cmd == "j" || cmd == "join":
3286         gshCtx.Rjoin(argv)
3287     case cmd == "a" || cmd == "alpa":
3288         gshCtx.Rexec(argv)
3289     case cmd == "jcd" || cmd == "jchdir":
3290         gshCtx.Rchdir(argv)
3291     case cmd == "jget":
3292         gshCtx.Rget(argv)
3293     case cmd == "jls":
3294         gshCtx.Rls(argv)
3295     case cmd == "jput":
3296         gshCtx.Rput(argv)
3297     case cmd == "jpwd":
3298         gshCtx.Rpwd(argv)
3299
3300     case cmd == "time":
3301         fin = gshCtx.xTime(argv)
3302     case cmd == "ungets":
3303         if 1 < len(argv) {
3304             ungets(argv[1]+\n")
3305         }else{
3306         }
3307     case cmd == "pwd":
3308         gshCtx.xPwd(argv);
3309     case cmd == "ver" || cmd == "-ver" || cmd == "version":
3310         gshCtx.showVersion(argv)
3311     case cmd == "where":
3312         // data file or so?
3313     case cmd == "which":
3314         which("PATH",argv);
3315     default:
3316         if gshCtx.whichPlugin(cmd,[]string{"-s"}) != nil {
3317             gshCtx.xPlugin(argv)
3318         }else{
3319             notfound,_ := gshCtx.excommand(false,argv)
3320             if notfound {
3321                 fmt.Printf("--E-- command not found (%v)\n",cmd)
3322             }
3323         }
3324     }
3325     return fin
3326 }
3327
3328 func (gsh*GshContext)gshell(gline string) (rfin bool) {
3329     argv := strings.Split(string(gline), " ")
3330     fin := gsh.gshellv(argv)
3331     return fin
3332 }
3333 func (gsh*GshContext)tgshell(gline string)(xfin bool){
3334     start := time.Now()
3335     fin := gsh.gshell(gline)
3336     end := time.Now()
3337     elps := end.Sub(start);
3338     if gsh.CmdTime {
3339         fmt.Printf("--T-- " + time.Now().Format(time.Stamp) + " (%d.%09ds)\n",
3340             elps/1000000000,elps*1000000000)
3341     }
3342     return fin
3343 }
3344 func Ttyid() (int) {
3345     fi, err := os.Stdin.Stat()
3346     if err != nil {
3347         return 0;
3348     }
3349     //fmt.Printf("Stdin: %v Dev=%d\n",
3350     // fi.Mode(),fi.Mode()&os.ModeDevice)
3351     if (fi.Mode() & os.ModeDevice) != 0 {
3352         stat := syscall.Stat_t{};
3353         err := syscall.Fstat(0,&stat)
3354         if err != nil {
3355             //fmt.Printf("--I-- Stdin: (%v)\n",err)
3356         }else{
3357             //fmt.Printf("--I-- Stdin: rdev=%d %d\n",
3358             // stat.Rdev&0xFF,stat.Rdev);
3359             //fmt.Printf("--I-- Stdin: tty%d\n",stat.Rdev&0xFF);
3360             return int(stat.Rdev & 0xFF)
3361         }
3362     }
3363     return 0
3364 }
3365 func (gshCtx *GshContext) ttyfile() string {
3366     //fmt.Printf("--I-- GSH_HOME=%s\n",gshCtx.GshHomeDir)
3367     ttyfile := gshCtx.GshHomeDir + "/" + "gsh-tty" +
3368         fmt.Sprintf("%02d",gshCtx.TerminalId)
3369     //strconv.Itoa(gshCtx.TerminalId)
3370     //fmt.Printf("--I-- ttyfile=%s\n",ttyfile)
3371     return ttyfile
3372 }
3373 func (gshCtx *GshContext) ttyline()(*os.File){
3374     file, err := os.OpenFile(gshCtx.ttyfile(),os.O_RDWR|os.O_CREATE|os.O_TRUNC,0600)

```

```

3375     if err != nil {
3376         fmt.Printf("--F-- cannot open %s (%s)\n", gshCtx.ttyfile(),err)
3377         return file;
3378     }
3379     return file
3380 }
3381 func (gshCtx *GshContext)getline(hix int, skipping bool, prevline string) (string) {
3382     if( skipping ){
3383         reader := bufio.NewReaderSize(os.Stdin,LINESIZE)
3384         line, _ ,_ = reader.ReadLine()
3385         return string(line)
3386     }else
3387     if true {
3388         return xgetline(hix,prevline,gshCtx)
3389     }/*
3390     else
3391     if( with_xgetline && gshCtx.GetLine != "" ){
3392         //var xhix int64 = int64(hix); // cast
3393         newenv := os.Environ()
3394         newenv = append(newenv, "GSH_LINENO="+strconv.FormatInt(int64(hix),10) )
3395
3396         tty := gshCtx.ttyline()
3397         tty.WriteString(prevline)
3398         Pa := os.ProcAttr {
3399             "", // start dir
3400             newenv, //os.Environ(),
3401             []*os.File{os.Stdin,os.Stdout,os.Stderr,tty},
3402             nil,
3403         }
3404     //fmt.Printf("--I-- getline=%s // %s\n",gsh_getlinev[0],gshCtx.GetLine)
3405     proc, err := os.StartProcess(gsh_getlinev[0],[]string{"getline","getline"},&Pa)
3406     if err != nil {
3407         fmt.Printf("--F-- getline process error (%v)\n",err)
3408         // for ; ; { }
3409         return "exit (getline program failed)"
3410     }
3411     //stat, err := proc.Wait()
3412     proc.Wait()
3413     buff := make([]byte,LINESIZE)
3414     count, err := tty.Read(buff)
3415     //_, err = tty.Read(buff)
3416     //fmt.Printf("--D-- getline (%d)\n",count)
3417     if err != nil {
3418         if ! (count == 0) { // && err.String() == "EOF" ) {
3419             fmt.Printf("--E-- getline error (%s)\n",err)
3420         }
3421     }else{
3422         //fmt.Printf("--I-- getline OK \"%s\"\n",buff)
3423     }
3424     tty.Close()
3425     gline := string(buff[0:count])
3426     return gline
3427 }
3428 */
3429 {
3430     // if isatty {
3431         fmt.Printf("!%d",hix)
3432         fmt.Println(PROMPT)
3433     //}
3434     reader := bufio.NewReaderSize(os.Stdin,LINESIZE)
3435     line, _ ,_ = reader.ReadLine()
3436     return string(line)
3437 }
3438 }
3439 */
3440 /**
3441 * getline.c
3442 * 2020-0819 extracted from dog.c
3443 * getline.go
3444 * 2020-0822 ported to Go
3445 */
3446 /*
3447 package main // getline main
3448 import (
3449     "fmt"          // <a href="https://golang.org/pkg/fmt/">fmt</a>
3450     "strings"       // <a href="https://golang.org/pkg/strings/">strings</a>
3451     "os"           // <a href="https://golang.org/pkg/os/">os</a>
3452     "syscall"      // <a href="https://golang.org/pkg/syscall/">syscall</a>
3453     //"bytes"      // <a href="https://golang.org/pkg/os/">os</a>
3454     //"os/exec"    // <a href="https://golang.org/pkg/os/">os</a>
3455 )
3456 */
3457
3458 // C language compatibility functions
3459 var errno = 0
3460 var stdin *os.File = os.Stdin
3461 var stdout *os.File = os.Stdout
3462 var stderr *os.File = os.Stderr
3463 var EOF = -1
3464 var NULL = 0
3465 type FILE os.File
3466 type StrBuff []byte
3467 var NULL_FPF *os.File = nil
3468 var NULLSP = 0
3469 //var LINESIZE = 1024
3470
3471 /**
3472 func system(cmdstr string)(int{
3473     PA := syscall.ProcAttr {
3474         "", // the starting directory
3475         os.Environ(),
3476         []uintptr{os.Stdin.Fd(),os.Stdout.Fd(),os.Stderr.Fd()},
3477         nil,
3478     }
3479     argv := strings.Split(cmdstr, " ")
3480     pid,err := syscall.ForkExec(argv[0],argv,&PA)
3481     if( err != nil ){
3482         fmt.Printf("--E-- syscall(%v) err(%v)\n",cmdstr,err)
3483     }
3484     syscall.Wait4(pid,nil,0,nil)
3485
3486     /*
3487     argv := strings.Split(cmdstr, " ")
3488     fmt.Fprintf(os.Stderr,"--I-- system(%v)\n",argv)
3489     //cmd := exec.Command(argv[0]...)
3490     cmd := exec.Command(argv[0],argv[1],argv[2])
3491     cmd.Stdin = strings.NewReader("output of system")
3492     var out bytes.Buffer
3493     cmd.Stdout = &out
3494     var serr bytes.Buffer
3495     cmd.Stderr = &serr
3496     err := cmd.Run()
3497     if err != nil {
3498         fmt.Fprintf(os.Stderr,"--E-- system(%v)err(%v)\n",argv,err)
3499     }

```

```
3500     fmt.Printf("ERR:%s\n",serr.String())
3501 }else{
3502     fmt.Printf("%s",out.String())
3503 }
3504 */
3505 return 0
3506 }
3507 func atoi(str string)(ret int){
3508     ret,err := fmt.Sscanf(str,"%d",ret)
3509     if err == nil {
3510         return ret
3511     }else{
3512         // should set errno
3513         return 0
3514     }
3515 }
3516 func getenv(name string)(string){
3517     val,got := os.LookupEnv(name)
3518     if got {
3519         return val
3520     }else{
3521         return "?"
3522     }
3523 }
3524 func strcpy(dst StrBuff, src string){
3525     var i int
3526     srcb := []byte(src)
3527     for i = 0; i < len(src) && srcb[i] != 0; i++ {
3528         dst[i] = srcb[i]
3529     }
3530     dst[i] = 0
3531 }
3532 func xstrcpy(dst StrBuff, src StrBuff){
3533     dst = src
3534 }
3535 func strcat(dst StrBuff, src StrBuff){
3536     dst = append(dst,src...)
3537 }
3538 func strdup(str StrBuff)(string){
3539     return string(str[:strlen(str)])
3540 }
3541 func strlen(str string)(int){
3542     return len(str)
3543 }
3544 func strlen(str StrBuff)(int){
3545     var i int
3546     for i = 0; i < len(str) && str[i] != 0; i++ {
3547     }
3548     return i
3549 }
3550 func sizeof(data StrBuff)(int){
3551     return len(data)
3552 }
3553 func isatty(fd int)(ret int){
3554     return 1
3555 }
3556
3557 func fopen(file string,mode string)(fp*os.File){
3558     if mode == "r" {
3559         fp,err := os.Open(file)
3560         if( err != nil ){
3561             fmt.Printf("--E-- fopen(%s,%s)=(%v)\n",file,mode,err)
3562             return NULL_FP;
3563         }
3564         return fp;
3565     }else{
3566         fp,err := os.OpenFile(file,os.O_RDWR|os.O_CREATE|os.O_TRUNC,0600)
3567         if( err != nil ){
3568             return NULL_FP;
3569         }
3570         return fp;
3571     }
3572 }
3573 func fclose(fp*os.File){
3574     fp.Close()
3575 }
3576 func fflush(fp *os.File)(int){
3577     return 0
3578 }
3579 func fgetc(fp*os.File)(int){
3580     var buf [1]byte
3581     ,err := fp.Read(buf[0:1])
3582     if( err != nil ){
3583         return EOF;
3584     }else{
3585         return int(buf[0])
3586     }
3587 }
3588 func sfgets(str*string, size int, fp*os.File)(int{
3589     buf := make(StrBuff,size)
3590     var ch int
3591     var i int
3592     for i = 0; i < len(buf)-1; i++ {
3593         ch = fgetc(fp)
3594         //fprintf(stderr,"--fgets %d/%d %x\n",i,len(buf),ch)
3595         if( ch == EOF ){
3596             break;
3597         }
3598         buf[i] = byte(ch);
3599         if( ch == '\n' ){
3600             break;
3601         }
3602     }
3603     buf[i] = 0
3604     //fprintf(stderr,"--fgets %d/%d (%s)\n",i,len(buf),buf[0:i])
3605     return i
3606 }
3607 func fgets(buf StrBuff, size int, fp*os.File)(int{
3608     var ch int
3609     var i int
3610     for i = 0; i < len(buf)-1; i++ {
3611         ch = fgetc(fp)
3612         //fprintf(stderr,"--fgets %d/%d %x\n",i,len(buf),ch)
3613         if( ch == EOF ){
3614             break;
3615         }
3616         buf[i] = byte(ch);
3617         if( ch == '\n' ){
3618             break;
3619         }
3620     }
3621     buf[i] = 0
3622     //fprintf(stderr,"--fgets %d/%d (%s)\n",i,len(buf),buf[0:i])
3623     return i
3624 }
```

```

3625 func fputc(ch int , fp*os.File)(int){
3626     var buf [1]byte
3627     buf[0] = byte(ch)
3628     fp.Write(buf[0:1])
3629     return 0
3630 }
3631 func fputs(buf StrBuff, fp*os.File)(int){
3632     fp.Write(buf)
3633     return 0
3634 }
3635 func xfputss(str string, fp*os.File)(int){
3636     return fputs([]byte(str),fp)
3637 }
3638 func scanf(str StrBuff,fmts string, params ...interface{})(int){
3639     fmt.Sscanf(string(str[0:strlen(str)]),fmts,params...)
3640     return 0
3641 }
3642 func fprintf(fp*os.File,fmts string, params ...interface{})(int){
3643     fmt.Fprintf(fp,fmts,params...)
3644     return 0
3645 }
3646
3647 //<a name="IME">Command Line IME</a>
3648 //----- MyIME
3649 var MyIMEVER = "MyIME/0.0.2";
3650 type RomKana struct {
3651     dic string // dictionary ID
3652     pat string // input pattern
3653     out string // output pattern
3654     hit int64 // count of hit and used
3655 }
3656 var dicensts = 0
3657 var romkana [1024]RomKana
3658 var Romkan []RomKana
3659
3660 func isinDic(str string)(int){
3661     for i,v := range Romkan {
3662         if v.pat == str {
3663             return i
3664         }
3665     }
3666     return -1
3667 }
3668 const (
3669     DIC_COM_LOAD = "im"
3670     DIC_COM_DUMP = "s"
3671     DIC_COM_LIST = "ls"
3672     DIC_COM_ENA = "en"
3673     DIC_COM_DIS = "di"
3674 )
3675 func helpDic(argv []string){
3676     out := stderr
3677     cmd := ""
3678     if 0 < len(argv) { cmd = argv[0] }
3679     printf(out,"--- %v Usage:\n",cmd)
3680     printf(out,"... Commands\n")
3681     printf(out,"... %v %-3v [dicName] [dicURL] -- Import dictionary\n",cmd,DIC_COM_LOAD)
3682     printf(out,"... %v %-3v [pattern] -- Search in dictionary\n",cmd,DIC_COM_DUMP)
3683     printf(out,"... %v %-3v [dicName] -- List dictionaries\n",cmd,DIC_COM_LIST)
3684     printf(out,"... %v %-3v [dicName] -- Disable dictionaries\n",cmd,DIC_COM_DIS)
3685     printf(out,"... %v %-3v [dicName] -- Enable dictionaries\n",cmd,DIC_COM_ENA)
3686     printf(out,"... Keys ... %v\n","ESC can be used for '\\\'')
3687     printf(out,"... \\c -- Reverse the case of the last character\n"),
3688     printf(out,"... \\i -- Replace input with translated text\n"),
3689     printf(out,"... \\j -- On/Off translation mode\n"),
3690     printf(out,"... \\l -- Force Lower Case\n"),
3691     printf(out,"... \\u -- Force Upper Case (software CapsLock)\n"),
3692     printf(out,"... \\v -- Show translation actions\n"),
3693     printf(out,"... \\x -- Replace the last input character with it Hexa-Decimal\n"),
3694 }
3695 func xdic(argv[]string){
3696     if len(argv) <= 1 {
3697         helpDic(argv)
3698         return
3699     }
3700     argv = argv[1:]
3701     var debug = false
3702     var info = false
3703     var silent = false
3704     var dump = false
3705     var builtin = false
3706     cmd := argv[0]
3707     argv = argv[1:]
3708     opt := ""
3709     arg := ""
3710
3711     if 0 < len(argv) {
3712         arg1 := argv[0]
3713         if arg1[0] == '-' {
3714             switch arg1 {
3715                 default: fmt.Printf("==Ed-- Unknown option(%v)\n",arg1)
3716                 return
3717                 case "-b": builtin = true
3718                 case "-d": debug = true
3719                 case "-s": silent = true
3720                 case "-v": info = true
3721             }
3722             opt = arg1
3723             argv = argv[1:]
3724         }
3725     }
3726
3727     dicName := ""
3728     dicURL := ""
3729     if 0 < len(argv) {
3730         arg = argv[0]
3731         dicName = arg
3732         argv = argv[1:]
3733     }
3734     if 0 < len(argv) {
3735         dicURL = argv[0]
3736         argv = argv[1:]
3737     }
3738     if false {
3739         fprintf(stderr,"==Dd-- com(%v) opt(%v) arg(%v)\n",cmd,opt,arg)
3740     }
3741     if cmd == DIC_COM_LOAD {
3742         //dicType := ""
3743         dicBody := ""
3744         if !builtin && dicName != "" && dicURL == "" {
3745             f,err := os.Open(dicName)
3746             if err == nil {
3747                 dicURL = dicName
3748             }else{
3749                 f,err = os.Open(dicName+".html")
3750             }
3751         }
3752         if !builtin && dicName != "" && dicURL != "" {
3753             f,err := os.Open(dicURL)
3754             if err == nil {
3755                 dicBody = f.ReadToString()
3756             }
3757         }
3758         if !builtin && dicBody != "" {
3759             f,err := os.Create(dicName)
3760             if err == nil {
3761                 f.Write(dicBody)
3762             }
3763         }
3764     }
3765     if !builtin && dicName != "" && dicURL != "" {
3766         f,err := os.Create(dicName)
3767         if err == nil {
3768             f.Write(dicBody)
3769         }
3770     }
3771     if !builtin && dicName == "" && dicURL == "" {
3772         f,err := os.Create(dicName)
3773         if err == nil {
3774             f.Write(dicBody)
3775         }
3776     }
3777     if !builtin && dicName == "" && dicURL != "" {
3778         f,err := os.Create(dicName)
3779         if err == nil {
3780             f.Write(dicBody)
3781         }
3782     }
3783     if !builtin && dicName != "" && dicURL == "" {
3784         f,err := os.Create(dicName)
3785         if err == nil {
3786             f.Write(dicBody)
3787         }
3788     }
3789     if !builtin && dicName != "" && dicURL != "" {
3790         f,err := os.Create(dicName)
3791         if err == nil {
3792             f.Write(dicBody)
3793         }
3794     }
3795     if !builtin && dicName == "" && dicURL == "" {
3796         f,err := os.Create(dicName)
3797         if err == nil {
3798             f.Write(dicBody)
3799         }
3800     }
3801     if !builtin && dicName != "" && dicURL != "" {
3802         f,err := os.Create(dicName)
3803         if err == nil {
3804             f.Write(dicBody)
3805         }
3806     }
3807     if !builtin && dicName == "" && dicURL == "" {
3808         f,err := os.Create(dicName)
3809         if err == nil {
3810             f.Write(dicBody)
3811         }
3812     }
3813     if !builtin && dicName != "" && dicURL != "" {
3814         f,err := os.Create(dicName)
3815         if err == nil {
3816             f.Write(dicBody)
3817         }
3818     }
3819     if !builtin && dicName == "" && dicURL == "" {
3820         f,err := os.Create(dicName)
3821         if err == nil {
3822             f.Write(dicBody)
3823         }
3824     }
3825     if !builtin && dicName != "" && dicURL != "" {
3826         f,err := os.Create(dicName)
3827         if err == nil {
3828             f.Write(dicBody)
3829         }
3830     }
3831     if !builtin && dicName == "" && dicURL == "" {
3832         f,err := os.Create(dicName)
3833         if err == nil {
3834             f.Write(dicBody)
3835         }
3836     }
3837     if !builtin && dicName != "" && dicURL != "" {
3838         f,err := os.Create(dicName)
3839         if err == nil {
3840             f.Write(dicBody)
3841         }
3842     }
3843     if !builtin && dicName == "" && dicURL == "" {
3844         f,err := os.Create(dicName)
3845         if err == nil {
3846             f.Write(dicBody)
3847         }
3848     }
3849     if !builtin && dicName != "" && dicURL != "" {
3850         f,err := os.Create(dicName)
3851         if err == nil {
3852             f.Write(dicBody)
3853         }
3854     }
3855     if !builtin && dicName == "" && dicURL == "" {
3856         f,err := os.Create(dicName)
3857         if err == nil {
3858             f.Write(dicBody)
3859         }
3860     }
3861     if !builtin && dicName != "" && dicURL != "" {
3862         f,err := os.Create(dicName)
3863         if err == nil {
3864             f.Write(dicBody)
3865         }
3866     }
3867     if !builtin && dicName == "" && dicURL == "" {
3868         f,err := os.Create(dicName)
3869         if err == nil {
3870             f.Write(dicBody)
3871         }
3872     }
3873     if !builtin && dicName != "" && dicURL != "" {
3874         f,err := os.Create(dicName)
3875         if err == nil {
3876             f.Write(dicBody)
3877         }
3878     }
3879     if !builtin && dicName == "" && dicURL == "" {
3880         f,err := os.Create(dicName)
3881         if err == nil {
3882             f.Write(dicBody)
3883         }
3884     }
3885     if !builtin && dicName != "" && dicURL != "" {
3886         f,err := os.Create(dicName)
3887         if err == nil {
3888             f.Write(dicBody)
3889         }
3890     }
3891     if !builtin && dicName == "" && dicURL == "" {
3892         f,err := os.Create(dicName)
3893         if err == nil {
3894             f.Write(dicBody)
3895         }
3896     }
3897     if !builtin && dicName != "" && dicURL != "" {
3898         f,err := os.Create(dicName)
3899         if err == nil {
3900             f.Write(dicBody)
3901         }
3902     }
3903     if !builtin && dicName == "" && dicURL == "" {
3904         f,err := os.Create(dicName)
3905         if err == nil {
3906             f.Write(dicBody)
3907         }
3908     }
3909     if !builtin && dicName != "" && dicURL != "" {
3910         f,err := os.Create(dicName)
3911         if err == nil {
3912             f.Write(dicBody)
3913         }
3914     }
3915     if !builtin && dicName == "" && dicURL == "" {
3916         f,err := os.Create(dicName)
3917         if err == nil {
3918             f.Write(dicBody)
3919         }
3920     }
3921     if !builtin && dicName != "" && dicURL != "" {
3922         f,err := os.Create(dicName)
3923         if err == nil {
3924             f.Write(dicBody)
3925         }
3926     }
3927     if !builtin && dicName == "" && dicURL == "" {
3928         f,err := os.Create(dicName)
3929         if err == nil {
3930             f.Write(dicBody)
3931         }
3932     }
3933     if !builtin && dicName != "" && dicURL != "" {
3934         f,err := os.Create(dicName)
3935         if err == nil {
3936             f.Write(dicBody)
3937         }
3938     }
3939     if !builtin && dicName == "" && dicURL == "" {
3940         f,err := os.Create(dicName)
3941         if err == nil {
3942             f.Write(dicBody)
3943         }
3944     }
3945     if !builtin && dicName != "" && dicURL != "" {
3946         f,err := os.Create(dicName)
3947         if err == nil {
3948             f.Write(dicBody)
3949         }
3950     }
3951     if !builtin && dicName == "" && dicURL == "" {
3952         f,err := os.Create(dicName)
3953         if err == nil {
3954             f.Write(dicBody)
3955         }
3956     }
3957     if !builtin && dicName != "" && dicURL != "" {
3958         f,err := os.Create(dicName)
3959         if err == nil {
3960             f.Write(dicBody)
3961         }
3962     }
3963     if !builtin && dicName == "" && dicURL == "" {
3964         f,err := os.Create(dicName)
3965         if err == nil {
3966             f.Write(dicBody)
3967         }
3968     }
3969     if !builtin && dicName != "" && dicURL != "" {
3970         f,err := os.Create(dicName)
3971         if err == nil {
3972             f.Write(dicBody)
3973         }
3974     }
3975     if !builtin && dicName == "" && dicURL == "" {
3976         f,err := os.Create(dicName)
3977         if err == nil {
3978             f.Write(dicBody)
3979         }
3980     }
3981     if !builtin && dicName != "" && dicURL != "" {
3982         f,err := os.Create(dicName)
3983         if err == nil {
3984             f.Write(dicBody)
3985         }
3986     }
3987     if !builtin && dicName == "" && dicURL == "" {
3988         f,err := os.Create(dicName)
3989         if err == nil {
3990             f.Write(dicBody)
3991         }
3992     }
3993     if !builtin && dicName != "" && dicURL != "" {
3994         f,err := os.Create(dicName)
3995         if err == nil {
3996             f.Write(dicBody)
3997         }
3998     }
3999     if !builtin && dicName == "" && dicURL == "" {
4000         f,err := os.Create(dicName)
4001         if err == nil {
4002             f.Write(dicBody)
4003         }
4004     }
4005     if !builtin && dicName != "" && dicURL != "" {
4006         f,err := os.Create(dicName)
4007         if err == nil {
4008             f.Write(dicBody)
4009         }
4010     }
4011     if !builtin && dicName == "" && dicURL == "" {
4012         f,err := os.Create(dicName)
4013         if err == nil {
4014             f.Write(dicBody)
4015         }
4016     }
4017     if !builtin && dicName != "" && dicURL != "" {
4018         f,err := os.Create(dicName)
4019         if err == nil {
4020             f.Write(dicBody)
4021         }
4022     }
4023     if !builtin && dicName == "" && dicURL == "" {
4024         f,err := os.Create(dicName)
4025         if err == nil {
4026             f.Write(dicBody)
4027         }
4028     }
4029     if !builtin && dicName != "" && dicURL != "" {
4030         f,err := os.Create(dicName)
4031         if err == nil {
4032             f.Write(dicBody)
4033         }
4034     }
4035     if !builtin && dicName == "" && dicURL == "" {
4036         f,err := os.Create(dicName)
4037         if err == nil {
4038             f.Write(dicBody)
4039         }
4040     }
4041     if !builtin && dicName != "" && dicURL != "" {
4042         f,err := os.Create(dicName)
4043         if err == nil {
4044             f.Write(dicBody)
4045         }
4046     }
4047     if !builtin && dicName == "" && dicURL == "" {
4048         f,err := os.Create(dicName)
4049         if err == nil {
4050             f.Write(dicBody)
4051         }
4052     }
4053     if !builtin && dicName != "" && dicURL != "" {
4054         f,err := os.Create(dicName)
4055         if err == nil {
4056             f.Write(dicBody)
4057         }
4058     }
4059     if !builtin && dicName == "" && dicURL == "" {
4060         f,err := os.Create(dicName)
4061         if err == nil {
4062             f.Write(dicBody)
4063         }
4064     }
4065     if !builtin && dicName != "" && dicURL != "" {
4066         f,err := os.Create(dicName)
4067         if err == nil {
4068             f.Write(dicBody)
4069         }
4070     }
4071     if !builtin && dicName == "" && dicURL == "" {
4072         f,err := os.Create(dicName)
4073         if err == nil {
4074             f.Write(dicBody)
4075         }
4076     }
4077     if !builtin && dicName != "" && dicURL != "" {
4078         f,err := os.Create(dicName)
4079         if err == nil {
4080             f.Write(dicBody)
4081         }
4082     }
4083     if !builtin && dicName == "" && dicURL == "" {
4084         f,err := os.Create(dicName)
4085         if err == nil {
4086             f.Write(dicBody)
4087         }
4088     }
4089     if !builtin && dicName != "" && dicURL != "" {
4090         f,err := os.Create(dicName)
4091         if err == nil {
4092             f.Write(dicBody)
4093         }
4094     }
4095     if !builtin && dicName == "" && dicURL == "" {
4096         f,err := os.Create(dicName)
4097         if err == nil {
4098             f.Write(dicBody)
4099         }
4100     }
4101     if !builtin && dicName != "" && dicURL != "" {
4102         f,err := os.Create(dicName)
4103         if err == nil {
4104             f.Write(dicBody)
4105         }
4106     }
4107     if !builtin && dicName == "" && dicURL == "" {
4108         f,err := os.Create(dicName)
4109         if err == nil {
4110             f.Write(dicBody)
4111         }
4112     }
4113     if !builtin && dicName != "" && dicURL != "" {
4114         f,err := os.Create(dicName)
4115         if err == nil {
4116             f.Write(dicBody)
4117         }
4118     }
4119     if !builtin && dicName == "" && dicURL == "" {
4120         f,err := os.Create(dicName)
4121         if err == nil {
4122             f.Write(dicBody)
4123         }
4124     }
4125     if !builtin && dicName != "" && dicURL != "" {
4126         f,err := os.Create(dicName)
4127         if err == nil {
4128             f.Write(dicBody)
4129         }
4130     }
4131     if !builtin && dicName == "" && dicURL == "" {
4132         f,err := os.Create(dicName)
4133         if err == nil {
4134             f.Write(dicBody)
4135         }
4136     }
4137     if !builtin && dicName != "" && dicURL != "" {
4138         f,err := os.Create(dicName)
4139         if err == nil {
4140             f.Write(dicBody)
4141         }
4142     }
4143     if !builtin && dicName == "" && dicURL == "" {
4144         f,err := os.Create(dicName)
4145         if err == nil {
4146             f.Write(dicBody)
4147         }
4148     }
4149     if !builtin && dicName != "" && dicURL != "" {
4150         f,err := os.Create(dicName)
4151         if err == nil {
4152             f.Write(dicBody)
4153         }
4154     }
4155     if !builtin && dicName == "" && dicURL == "" {
4156         f,err := os.Create(dicName)
4157         if err == nil {
4158             f.Write(dicBody)
4159         }
4160     }
4161     if !builtin && dicName != "" && dicURL != "" {
4162         f,err := os.Create(dicName)
4163         if err == nil {
4164             f.Write(dicBody)
4165         }
4166     }
4167     if !builtin && dicName == "" && dicURL == "" {
4168         f,err := os.Create(dicName)
4169         if err == nil {
4170             f.Write(dicBody)
4171         }
4172     }
4173     if !builtin && dicName != "" && dicURL != "" {
4174         f,err := os.Create(dicName)
4175         if err == nil {
4176             f.Write(dicBody)
4177         }
4178     }
4179     if !builtin && dicName == "" && dicURL == "" {
4180         f,err := os.Create(dicName)
4181         if err == nil {
4182             f.Write(dicBody)
4183         }
4184     }
4185     if !builtin && dicName != "" && dicURL != "" {
4186         f,err := os.Create(dicName)
4187         if err == nil {
4188             f.Write(dicBody)
4189         }
4190     }
4191     if !builtin && dicName == "" && dicURL == "" {
4192         f,err := os.Create(dicName)
4193         if err == nil {
4194             f.Write(dicBody)
4195         }
4196     }
4197     if !builtin && dicName != "" && dicURL != "" {
4198         f,err := os.Create(dicName)
4199         if err == nil {
4200             f.Write(dicBody)
4201         }
4202     }
4203     if !builtin && dicName == "" && dicURL == "" {
4204         f,err := os.Create(dicName)
4205         if err == nil {
4206             f.Write(dicBody)
4207         }
4208     }
4209     if !builtin && dicName != "" && dicURL != "" {
4210         f,err := os.Create(dicName)
4211         if err == nil {
4212             f.Write(dicBody)
4213         }
4214     }
4215     if !builtin && dicName == "" && dicURL == "" {
4216         f,err := os.Create(dicName)
4217         if err == nil {
4218             f.Write(dicBody)
4219         }
4220     }
4221     if !builtin && dicName != "" && dicURL != "" {
4222         f,err := os.Create(dicName)
4223         if err == nil {
4224             f.Write(dicBody)
4225         }
4226     }
4227     if !builtin && dicName == "" && dicURL == "" {
4228         f,err := os.Create(dicName)
4229         if err == nil {
4230             f.Write(dicBody)
4231         }
4232     }
4233     if !builtin && dicName != "" && dicURL != "" {
4234         f,err := os.Create(dicName)
4235         if err == nil {
4236             f.Write(dicBody)
4237         }
4238     }
4239     if !builtin && dicName == "" && dicURL == "" {
4240         f,err := os.Create(dicName)
4241         if err == nil {
4242             f.Write(dicBody)
4243         }
4244     }
4245     if !builtin && dicName != "" && dicURL != "" {
4246         f,err := os.Create(dicName)
4247         if err == nil {
4248             f.Write(dicBody)
4249         }
4250     }
4251     if !builtin && dicName == "" && dicURL == "" {
4252         f,err := os.Create(dicName)
4253         if err == nil {
4254             f.Write(dicBody)
4255         }
4256     }
4257     if !builtin && dicName != "" && dicURL != "" {
4258         f,err := os.Create(dicName)
4259         if err == nil {
4260             f.Write(dicBody)
4261         }
4262     }
4263     if !builtin && dicName == "" && dicURL == "" {
4264         f,err := os.Create(dicName)
4265         if err == nil {
4266             f.Write(dicBody)
4267         }
4268     }
4269     if !builtin && dicName != "" && dicURL != "" {
4270         f,err := os.Create(dicName)
4271         if err == nil {
4272             f.Write(dicBody)
4273         }
4274     }
4275     if !builtin && dicName == "" && dicURL == "" {
4276         f,err := os.Create(dicName)
4277         if err == nil {
4278             f.Write(dicBody)
4279         }
4280     }
4281     if !builtin && dicName != "" && dicURL != "" {
4282         f,err := os.Create(dicName)
4283         if err == nil {
4284             f.Write(dicBody)
4285         }
4286     }
4287     if !builtin && dicName == "" && dicURL == "" {
4288         f,err := os.Create(dicName)
4289         if err == nil {
4290             f.Write(dicBody)
4291         }
4292     }
4293     if !builtin && dicName != "" && dicURL != "" {
4294         f,err := os.Create(dicName)
4295         if err == nil {
4296             f.Write(dicBody)
4297         }
4298     }
4299     if !builtin && dicName == "" && dicURL == "" {
4300         f,err := os.Create(dicName)
4301         if err == nil {
4302             f.Write(dicBody)
4303         }
4304     }
4305     if !builtin && dicName != "" && dicURL != "" {
4306         f,err := os.Create(dicName)
4307         if err == nil {
4308             f.Write(dicBody)
4309         }
4310     }
4311     if !builtin && dicName == "" && dicURL == "" {
4312         f,err := os.Create(dicName)
4313         if err == nil {
4314             f.Write(dicBody)
4315         }
4316     }
4317     if !builtin && dicName != "" && dicURL != "" {
4318         f,err := os.Create(dicName)
4319         if err == nil {
4320             f.Write(dicBody)
4321         }
4322     }
4323     if !builtin && dicName == "" && dicURL == "" {
4324         f,err := os.Create(dicName)
4325         if err == nil {
4326             f.Write(dicBody)
4327         }
4328     }
4329     if !builtin && dicName != "" && dicURL != "" {
4330         f,err := os.Create(dicName)
4331         if err == nil {
4332             f.Write(dicBody)
4333         }
4334     }
4335     if !builtin && dicName == "" && dicURL == "" {
4336         f,err := os.Create(dicName)
4337         if err == nil {
4338             f.Write(dicBody)
4339         }
4340     }
4341     if !builtin && dicName != "" && dicURL != "" {
4342         f,err := os.Create(dicName)
4343         if err == nil {
4344             f.Write(dicBody)
4345         }
4346     }
4347     if !builtin && dicName == "" && dicURL == "" {
4348         f,err := os.Create(dicName)
4349         if err == nil {
4350             f.Write(dicBody)
4351         }
4352     }
4353     if !builtin && dicName != "" && dicURL != "" {
4354         f,err := os.Create(dicName)
4355         if err == nil {
4356             f.Write(dicBody)
4357         }
4358     }
4359     if !builtin && dicName == "" && dicURL == "" {
4360         f,err := os.Create(dicName)
4361         if err == nil {
4362             f.Write(dicBody)
4363         }
4364     }
4365     if !builtin && dicName != "" && dicURL != "" {
4366         f,err := os.Create(dicName)
4367         if err == nil {
4368             f.Write(dicBody)
4369         }
4370     }
4371     if !builtin && dicName == "" && dicURL == "" {
4372         f,err := os.Create(dicName)
4373         if err == nil {
4374             f.Write(dicBody)
4375         }
4376     }
4377     if !builtin && dicName != "" && dicURL != "" {
4378         f,err := os.Create(dicName)
4379         if err == nil {
4380             f.Write(dicBody)
4381         }
4382     }
4383     if !builtin && dicName == "" && dicURL == "" {
4384         f,err := os.Create(dicName)
4385         if err == nil {
4386             f.Write(dicBody)
4387         }
4388     }
4389     if !builtin && dicName != "" && dicURL != "" {
4390         f,err := os.Create(dicName)
4391         if err == nil {
4392             f.Write(dicBody)
4393         }
4394     }
4395     if !builtin && dicName == "" && dicURL == "" {
4396         f,err := os.Create(dicName)
4397         if err == nil {
4398             f.Write(dicBody)
4399         }
4400     }
4401     if !builtin && dicName != "" && dicURL != "" {
4402         f,err := os.Create(dicName)
4403         if err == nil {
4404             f.Write(dicBody)
4405         }
4406     }
4407     if !builtin && dicName == "" && dicURL == "" {
4408         f,err := os.Create(dicName)
4409         if err == nil {
4410             f.Write(dicBody)
4411         }
4412     }
4413     if !builtin && dicName != "" && dicURL != "" {
4414         f,err := os.Create(dicName)
4415         if err == nil {
4416             f.Write(dicBody)
4417         }
4418     }
4419     if !builtin && dicName == "" && dicURL == "" {
4420         f,err := os.Create(dicName)
4421         if err == nil {
4422             f.Write(dicBody)
4423         }
4424     }
4425     if !builtin && dicName != "" && dicURL != "" {
4426         f,err := os.Create(dicName)
4427         if err == nil {
4428             f.Write(dicBody)
4429         }
4430     }
4431     if !builtin && dicName == "" && dicURL == "" {
4432         f,err := os.Create(dicName)
4433         if err == nil {
4434             f.Write(dicBody)
4435         }
4436     }
4437     if !builtin && dicName != "" && dicURL != "" {
4438         f,err := os.Create(dicName)
4439         if err == nil {
4440             f.Write(dicBody)
4441         }
4442     }
4443     if !builtin && dicName == "" && dicURL == "" {
4444         f,err := os.Create(dicName)
4445         if err == nil {
4446             f.Write(dicBody)
4447         }
4448     }
4449     if !builtin && dicName != "" && dicURL != "" {
4450         f,err := os.Create(dicName)
4451         if err == nil {
4452             f.Write(dicBody)
4453         }
4454     }
4455     if !builtin && dicName == "" && dicURL == "" {
4456         f,err := os.Create(dicName)
4457         if err == nil {
4458             f.Write(dicBody)
4459         }
4460     }
4461     if !builtin && dicName != "" && dicURL != "" {
4462         f,err := os.Create(dicName)
4463         if err == nil {
4464             f.Write(dicBody)
4465         }
4466     }
4467     if !builtin && dicName == "" && dicURL == "" {
4468         f,err := os.Create(dicName)
4469         if err == nil {
4470             f.Write(dicBody)
4471         }
4472     }
4473     if !builtin && dicName != "" && dicURL != "" {
4474         f,err := os.Create(dicName)
4475         if err == nil {
4476             f.Write(dicBody)
4477         }
4478     }
4479     if !builtin && dicName == "" && dicURL == "" {
4480         f,err := os.Create(dicName)
4481         if err == nil {
4482             f.Write(dicBody)
4483         }
4484     }
4485     if !builtin && dicName != "" && dicURL != "" {
4486         f,err := os.Create(dicName)
4487         if err == nil {
4488             f.Write(dicBody)
4489         }
4490     }
4491     if !builtin && dicName == "" && dicURL == "" {
4492         f,err := os.Create(dicName)
4493         if err == nil {
4494             f.Write(dicBody)
4495         }
4496     }
4497     if !builtin && dicName != "" && dicURL != "" {
4498         f,err := os.Create(dicName)
4499         if err == nil {
4500             f.Write(dicBody)
4501         }
4502     }
4503     if !builtin && dicName == "" && dicURL == "" {
4504         f,err := os.Create(dicName)
4505         if err == nil {
4506             f.Write(dicBody)
4507         }
4508     }
4509     if !builtin && dicName != "" && dicURL != "" {
4510         f,err := os.Create(dicName)
4511         if err == nil {
4512             f.Write(dicBody)
4513         }
4514     }
4515     if !builtin && dicName == "" && dicURL == "" {
4516         f,err := os.Create(dicName)
4517         if err == nil {
4518             f.Write(dicBody)
4519         }
4520     }
4521     if !builtin && dicName != "" && dicURL != "" {
4522         f,err := os.Create(dicName)
4523         if err == nil {
4524             f.Write(dicBody)
4525         }
4526     }
4527     if !builtin && dicName == "" && dicURL == "" {
4528         f,err := os.Create(dicName)
4529         if err == nil {
4530             f.Write(dicBody)
4531         }
4532     }
4533     if !builtin && dicName != "" && dicURL != "" {
4534         f,err := os.Create(dicName)
4535         if err == nil {
4536             f.Write(dicBody)
4537         }
4538     }
4539     if !builtin && dicName == "" && dicURL == "" {
4540         f,err := os.Create(dicName)
4541         if err == nil {
4542             f.Write(dicBody)
45
```

```

3750     if err == nil {
3751         dicURL = dicName+".html"
3752     }else{
3753         f,err = os.Open("gshdic-"+dicName+".html")
3754         if err == nil {
3755             dicURL = "gshdic-"+dicName+".html"
3756         }
3757     }
3758 }
3759 if err == nil {
3760     var buf = make([]byte,128*1024)
3761     count,err := f.Read(buf)
3762     f.Close()
3763     if info {
3764         fprintf(stderr,"--Id-- ReadDic(%v,%v)\n",count,err)
3765     }
3766     dicBody = string(buf[0:count])
3767 }
3768 if dicBody == "" {
3769     switch arg {
3770     default:
3771         dicName = "WorldDic"
3772         dicURL = WorldDic
3773         if info {
3774             fprintf(stderr,"--Id-- default dictionary \"%v\"\n",
3775                     dicName);
3776         }
3777     case "wnn":
3778         dicName = "WnnDic"
3779         dicURL = WnnDic
3780     case "sumomo":
3781         dicName = "SumomoDic"
3782         dicURL = SumomoDic
3783     case "sijimi":
3784         dicName = "SijimiDic"
3785         dicURL = SijimiDic
3786     case "jkl":
3787         dicName = "JKLJaDic"
3788         dicURL = JA_JKLDic
3789     }
3790 }
3791 if debug {
3792     fprintf(stderr,"--Id-- %v URL=%v\n",dicName,dicURL);
3793 }
3794 dicv := strings.Split(dicURL,",")
3795 if debug {
3796     fprintf(stderr,"--Id-- %v encoded data...\n",dicName)
3797     fprintf(stderr,"type: %v\n",dicv[0])
3798     fprintf(stderr,"Body: %v\n",dicv[1])
3799     fprintf(stderr,"\n")
3800 }
3801 body,_ := base64.StdEncoding.DecodeString(dicv[1])
3802 dicBody = string(body)
3803 }
3804 if info {
3805     fmt.Printf("--Id-- %v %v\n",dicName,dicURL)
3806     fmt.Printf("%s\n",dicBody)
3807 }
3808 if debug {
3809     fprintf(stderr,"--Id-- dicName %v text...\n",dicName)
3810     fprintf(stderr,"%v\n",string(dicBody))
3811 }
3812 envt := strings.Split(dicBody,"\n");
3813 if info {
3814     fprintf(stderr,"--Id-- %v scan...\n",dicName);
3815 }
3816 var added int = 0
3817 var dup int = 0
3818 for i,v := range envt {
3819     var pat string
3820     var out string
3821     fmt.Sscanf(v,"%s %s",&pat,&out)
3822     if len(pat) <= 0 {
3823         if 0 <= isinDic(pat) {
3824             dup += 1
3825             continue
3826         }
3827         romkana[dicents] = RomKana{dicName,pat,out,0}
3828         dicents += 1
3829         added += 1
3830         Romkan = append(Romkan,RomKana{dicName,pat,out,0})
3831         if debug {
3832             fmt.Printf("[%3v]:[%2v]%-8v [%2v]%v\n",
3833                         i,len(pat),pat,len(out),out)
3834         }
3835     }
3836 }
3837 if !silent {
3838     url := dicURL
3839     if strBegins(url,"data:") {
3840         url = "builtin"
3841     }
3842     fprintf(stderr,"--Id-- %v scan... %v added, %v dup. / %v total (%v)\n",
3843             dicName,added,dup,len(Romkan),url);
3844 }
3845 // should sort by pattern length for concrete match, for performance
3846 if debug {
3847     arg = "" // search pattern
3848     dump = true
3849 }
3850 }
3851 }
3852 if cmd == DIC_COM_DUMP || dump {
3853     fprintf(stderr,"--Id-- %v dump... %v entries:\n",dicName,len(Romkan));
3854     var match = 0
3855     for i := 0; i < len(Romkan); i++ {
3856         dic := Romkan[i].dic
3857         pat := Romkan[i].pat
3858         out := Romkan[i].out
3859         if arg == "" || 0 <= strings.Index(pat,arg)||0 <= strings.Index(out,arg) {
3860             fmt.Printf("\\\\%v\\%v [%2v]%-8v [%2v]%v\n",
3861                         i,dic,len(pat),pat,len(out),out)
3862             match += 1
3863         }
3864     }
3865     fprintf(stderr,"--Id-- %v matched %v / %v entries:\n",arg,match,len(Romkan));
3866 }
3867 }
3868 func loadDefaultDic(dic int){
3869     if( 0 < len(Romkan) ){
3870         return
3871     }
3872     //fprintf(stderr,"\\r\\n")
3873     xDic([]string{"dic",DIC_COM_LOAD});
3874 }
```

```

3875     var info = false
3876     if info {
3877         fprintf(stderr,"--Id-- Conguratulations!! WorldDic is now activated.\r\n")
3878         fprintf(stderr,"--Id-- enter \"dic\" command for help.\r\n")
3879     }
3880 }
3881 func readDic()(int){
3882     /*
3883     var rk *os.File;
3884     var dic = "MyIME-dic.txt";
3885     //rk = fopen("romkana.txt","r");
3886     //rk = fopen("JK-JA-morse-dic.txt","r");
3887     rk = fopen(dic,"r");
3888     if( rk == NULL_fp ){
3889         if( true ){
3890             fprintf(stderr,"--%s-- Could not load %s\n",MyIMEVER,dic);
3891         }
3892         return -1;
3893     }
3894     if( true ){
3895         var di int;
3896         var line = make(StrBuff,1024);
3897         var pat string
3898         var out string
3899         for di = 0; di < 1024; di++ {
3900             if( fgets(line,sizeof(line),rk) == NULLSP ){
3901                 break;
3902             }
3903             fmt.Sscanf(string(line[0:strlen(line)]),"s s",&pat,&out);
3904             //sscanf(line,"%s [%r\n]",&pat,&out);
3905             romkana[di].pat = pat;
3906             romkana[di].out = out;
3907             //fprintf(stderr,"--Dd- %-10s %s\n",pat,out)
3908         }
3909         dicents += di
3910         if( false ){
3911             fprintf(stderr,"--%s-- loaded romkana.txt [%d]\n",MyIMEVER,di);
3912             for di = 0; di < dicents; di++ {
3913                 fprintf(stderr,
3914                     "%s %s\n",romkana[di].pat,romkana[di].out);
3915             }
3916         }
3917     }
3918     fclose(rk);
3919
3920     //romkana[dicents].pat = "//ddump"
3921     //romkana[dicents].pat = "//ddump" // dump the dic. and clean the command input
3922 */
3923     return 0;
3924 }
3925 func matchlen(stri string, pati string)(int){
3926     if strBegins(stri,pati) {
3927         return len(pati)
3928     }else{
3929         return 0
3930     }
3931 }
3932 func convs(src string)(string){
3933     var si int;
3934     var sx = len(src);
3935     var di int;
3936     var mi int;
3937     var dstb []byte
3938
3939     for si = 0; si < sx; { // search max. match from the position
3940         if strBegins(src[si:], "%x") {
3941             // %x/integer/ // s/a/b/
3942             ix := strings.Index(src[si+3:], "/")
3943             if 0 < ix {
3944                 var iv int = 0
3945                 fmt.Sscanf(src[si+3:si+3+ix],"%d",&iv)
3946                 fmt.Sscanf(src[si+3:si+3+ix],"%v",&iv)
3947                 sval := fmt.Sprintf("%x",iv)
3948                 bval := []byte(sval)
3949                 dstb = append(dstb,bval...)
3950                 si = si+3+ix+1
3951                 continue
3952             }
3953             if strBegins(src[si:], "%d/") {
3954                 // %d/integer/ // s/a/b/
3955                 ix := strings.Index(src[si+3:], "/")
3956                 if 0 < ix {
3957                     var iv int = 0
3958                     fmt.Sscanf(src[si+3:si+3+ix],"%v",&iv)
3959                     sval := fmt.Sprintf("%d",iv)
3960                     bval := []byte(sval)
3961                     dstb = append(dstb,bval...)
3962                     si = si+3+ix+1
3963                     continue
3964                 }
3965             }
3966             if strBegins(src[si:], "%t") {
3967                 now := time.Now()
3968                 if true {
3969                     date := now.Format(time.Stamp)
3970                     dstb = append(dstb,[]byte(date)...),
3971                     si = si+3
3972                 }
3973                 continue
3974             }
3975             var maxlen int = 0;
3976             var len int;
3977             mi = -1;
3978             for di = 0; di < dicents; di++ {
3979                 len = matchlen(src[si:],romkana[di].pat);
3980                 if( maxlen < len ){
3981                     maxlen = len;
3982                     mi = di;
3983                 }
3984             }
3985             if( 0 < maxlen ){
3986                 out := romkana[mi].out;
3987                 dstb = append(dstb,[]byte(out)...),
3988                 si += maxlen;
3989             }else{
3990                 dstb = append(dstb,src[si])
3991                 si += 1;
3992             }
3993         }
3994     }
3995     return string(dstb)
3996 }
3997 func trans(src string)(int){
3998     dst := convs(src);
3999     xfputss(dst,stderr);

```

```

4000     return 0;
4001 }
4002
4003 //----- LINEEDIT
4004 // "?" at the top of the line means searching history
4005
4006 // should be compatilbe with Telnet
4007 const {
4008     EV_MODE      = 255
4009     EV_IDLE     = 254
4010     EV_TIMEOUT  = 253
4011
4012     GO_UP       = 252 // k
4013     GO_DOWN     = 251 // j
4014     GO_RIGHT    = 250 // l
4015     GO_LEFT     = 249 // h
4016     DEL_RIGHT   = 248 // x
4017     GO_TOPL     = 'A'-0x40 // 0
4018     GO_ENDL     = 'E'-0x40 // $
4019
4020     GO_TOPW     = 239 // b
4021     GO_ENDW     = 238 // e
4022     GO_NEXTW    = 237 // w
4023
4024     GO_FORWCH   = 229 // f
4025     GO_PAIRCH   = 228 // $
4026
4027     GO_DEL      = 219 // d
4028
4029     HI_SRCH_FW  = 209 // /
4030     HI_SRCH_BK  = 208 // ?
4031     HI_SRCH_RFW = 207 // n
4032     HI_SRCH_RBK = 206 // N
4033 }
4034
4035 // should return number of octets ready to be read immediately
4036 //fprintf(stderr,"\\n--Select(%v %v)\\n",err,r.Bits[0])
4037
4038
4039 var EventRecvFd = -1 // file descriptor
4040 var EventSendFd = -1
4041 const EventFdOffset = 1000000
4042 const NormalFdOffset = 100
4043
4044 func putEvent(event int, evarg int){
4045     if true {
4046         if EventRecvFd < 0 {
4047             var pv = []int{-1,-1}
4048             syscall.Pipe(pv)
4049             EventRecvFd = pv[0]
4050             EventSendFd = pv[1]
4051             //fmt.Printf("--De-- EventPipe created[%v,%v]\\n",EventRecvFd,EventSendFd)
4052         }
4053     }else{
4054         if EventRecvFd < 0 {
4055             // the document differs from this spec
4056             // https://golang.org/src/syscall/syscall_unix.go?s=8096:8158#L340
4057             sv,err := syscall.Socketpair(syscall.AF_UNIX,syscall.SOCK_STREAM,0)
4058             EventRecvFd = sv[0]
4059             EventSendFd = sv[1]
4060             if err != nil {
4061                 fmt.Printf("--De-- EventSock created[%v,%v](%v)\\n",
4062                     EventRecvFd,EventSendFd,err)
4063             }
4064         }
4065     }
4066     var buf = []byte{ byte(event) }
4067     n,err := syscall.Write(EventSendFd,buf)
4068     if err != nil {
4069         fmt.Printf("--De-- putEvent[%v](%v)(%v)\\n",EventSendFd,event,n,err)
4070     }
4071 }
4072 func ungets(str string){
4073     for _,ch := range str {
4074         putEvent(int(ch),0)
4075     }
4076 }
4077 func (gsh*GshContext)xReplay(argv[]string){
4078     hix := 0
4079     tempo := 1.0
4080     xtempo := 1.0
4081     repeat := 1
4082
4083     for _,a := range argv { // tempo
4084         if strBegins(a,"x") {
4085             fmt.Sscanf(a[1:], "%f", &xtempo)
4086             tempo = 1 / xtempo
4087             //fprintf(stderr,"--Dr-- tempo=[%v]\\n",a[2:],tempo);
4088         }else
4089         if strBegins(a,"r") { // repeat
4090             fmt.Sscanf(a[1:], "%v", &repeat)
4091         }else
4092         if strBegins(a,"!") {
4093             fmt.Sscanf(a[1:], "%d", &hix)
4094         }else{
4095             fmt.Sscanf(a,"%d", &hix)
4096         }
4097     }
4098     if hix == 0 || len(argv) <= 1 {
4099         hix = len(gsh.CommandHistory)-1
4100     }
4101     fmt.Printf("--Ir-- Replay(!%v x%v r%v)\\n",hix,xtempo,repeat)
4102     //dumpEvents(hix)
4103     //gsh.xScanReplay(hix,false,repeat,tempo,argv)
4104     go gsh.xScanReplay(hix,true,repeat,tempo,argv)
4105 }
4106
4107 // <a href="https://golang.org/pkg/syscall/#FdSet">syscall.Select</a>
4108 // 2020-0827 GShell-0.2.3
4109 /*
4110 func FpollInit(fp *os.File,usec int)(uintptr{
4111     nfd := 1
4112
4113     rdv := syscall.FdSet {}
4114     fd1 := fp.Fd()
4115     bank1 := fd1/32
4116     mask1 := int32(1 << fd1)
4117     rdv.Bits[bank1] = mask1
4118
4119     fd2 := -1
4120     bank2 := -1
4121     var mask2 int32 = 0
4122
4123     if 0 <= EventRecvFd {
4124         fd2 = EventRecvFd

```

```

4125     nfd = fd2 + 1
4126     bank2 = fd2/32
4127     mask2 = int32(1 << fd2)
4128     rdv.Bits[bank2] |= mask2
4129     //fmt.Printf("--De-- EventPoll mask added [%d][%v]\n",fd2,bank2,mask2)
4130 }
4131
4132 tout := syscall.NsecToTimeval(int64(usec*1000))
4133 //n,err := syscall.Select(nfd,&rdv,nil,nil,&tout) // spec. mismatch
4134 err := syscall.Select(nfd,&rdv,nil,nil,&tout)
4135 if err != nil {
4136     //fmt.Printf("--De-- select() err(%v)\n",err)
4137 }
4138 if err == nil {
4139     if 0 <= fd2 && (rdv.Bits[bank2] & mask2) != 0 {
4140         if false {
4141             fmt.Printf("--De-- got Event\n")
4142         }
4143         return uintptr(EventFdOffset + fd2)
4144     }else
4145     if (rdv.Bits[bank1] & mask1) != 0 {
4146         return uintptr(NormalFdOffset + fd1)
4147     }else{
4148         return 1
4149     }
4150 }else{
4151     return 0
4152 }
4153 */
4154 */
4155 func fgetcTimeout(fp *os.File,usec int)(int){
4156     READ1:
4157     //readyFd := FpollInl(fp,usec)
4158     readyFd := CFpollInl(fp,usec)
4159     if readyFd < 100 {
4160         return EV_TIMEOUT
4161     }
4162
4163     var buf [1]byte
4164
4165     if EventFdOffset <= readyFd {
4166         fd := int(readyFd-EventFdOffset)
4167         _,err := syscall.Read(fd,buf[0:1])
4168         if( err != nil ){
4169             return EOF;
4170         }else{
4171             if buf[0] == EV_MODE {
4172                 recvEvent(fd)
4173                 goto READ1
4174             }
4175             return int(buf[0])
4176         }
4177     }
4178
4179     _,err := fp.Read(buf[0:1])
4180     if( err != nil ){
4181         return EOF;
4182     }else{
4183         return int(buf[0])
4184     }
4185 }
4186
4187 func visibleChar(ch int)(string){
4188     switch {
4189         case '!' <= ch && ch <= '~':
4190             return string(ch)
4191     }
4192     switch ch {
4193         case ' ': return "\\s"
4194         case '\n': return "\\n"
4195         case '\r': return "\\r"
4196         case '\t': return "\\t"
4197     }
4198     switch ch {
4199         case 0x00: return "NUL"
4200         case 0x07: return "BEL"
4201         case 0x08: return "BS"
4202         case 0x0E: return "SO"
4203         case 0x0F: return "SI"
4204         case 0x1B: return "ESC"
4205         case 0x7F: return "DEL"
4206     }
4207     switch ch {
4208         case EV_IDLE: return fmt.Sprintf("IDLE")
4209         case EV_MODE: return fmt.Sprintf("MODE")
4210     }
4211     return fmt.Sprintf("%x",ch)
4212 }
4213 func recvEvent(fd int){
4214     var buf = make([]byte,1)
4215     _,_ = syscall.Read(fd,buf[0:1])
4216     if( buf[0] != 0 ){
4217         romkanmode = true
4218     }else{
4219         romkanmode = false
4220     }
4221 }
4222 func (gsh*GshContext)xScanReplay(hix int,replay bool,repeat int,tempo float64,argv[]string){
4223     var Start time.Time
4224     var events = []Event{}
4225     for e := range Events {
4226         if hix == 0 || e.CmdIndex == hix {
4227             events = append(events,e)
4228         }
4229     }
4230     elen := len(events)
4231     if 0 < elen {
4232         if events[elen-1].event == EV_IDLE {
4233             events = events[0:elen-1]
4234         }
4235     }
4236     for r := 0; r < repeat; r++ {
4237         for i,e := range events {
4238             nano := e.when.Nanosecond()
4239             micro := nano / 1000
4240             if Start.Second() == 0 {
4241                 Start = time.Now()
4242             }
4243             diff := time.Now().Sub(Start)
4244             if replay {
4245                 if e.event != EV_IDLE {
4246                     putEvent(e.event,0)
4247                     if e.event == EV_MODE { // event with arg
4248                         putEvent(int(e.evarg),0)
4249                     }
4250                 }
4251             }
4252         }
4253     }
4254 }

```

```

4250     }
4251 }else{
4252     fmt.Printf("%7.3fms #%-3v !%-3v [%v.%06d] %3v %02X %-4v %10.3fms\n",
4253         float64(diff)/1000000.0,
4254         i,
4255         e.CmdIndex,
4256         e.when.Format(time.Stamp),micro,
4257         e.event,e.event,visibleChar(e.event),
4258         float64(e.evarg)/1000000.0)
4259 }
4260 if e.event == EV_IDLE {
4261     d := time.Duration(float64(time.Duration(e.evarg)) * tempo)
4262     //nsleep(time.Duration(e.evarg))
4263     nsleep(d)
4264 }
4265 }
4266 }
4267 }
4268 func dumpEvents(arg[]string){
4269     hix := 0
4270     if 1 < len(arg) {
4271         fmt.Sscanf(arg[1],"%d",&hix)
4272     }
4273     for i,e := range Events {
4274         nano := e.when.Nanosecond()
4275         micro := nano / 1000
4276         //if e.event != EV_TIMEOUT {
4277         if hix == 0 || e.CmdIndex == hix {
4278             fmt.Printf("#%-3v !%-3v [%v.%06d] %3v %02X %-4v %10.3fms\n",i,
4279                 e.CmdIndex,
4280                 e.when.Format(time.Stamp),micro,
4281                 e.event,e.event,visibleChar(e.event),float64(e.evarg)/1000000.0)
4282         }
4283     }
4284 }
4285 }
4286 func fgetcTimeout(fp *os.File,usec int)(int){
4287     ch := fgetcTimeout(fp,usec)
4288     if ch != EV_TIMEOUT {
4289         now := time.Now()
4290         if 0 < len(Events) {
4291             last := Events[len(Events)-1]
4292             dura := int64(now.Sub(last.when))
4293             Events = append(Events,Event{last.when,EV_IDLE,dura,last.CmdIndex})
4294         }
4295         Events = append(Events,Event{time.Now(),ch,0,CmdIndex})
4296     }
4297     return ch
4298 }
4299 }
4300 var TtyMaxCol = 72 // to be obtained by ioctl?
4301 var EscTimeout = (100*1000)
4302 var {
4303     MODE_VicMode    bool    // vi compatible command mode
4304     MODE_ShowMode   bool
4305     romkammode bool    // shown translation mode, the mode to be retained
4306     MODE_Recursive  bool    // recursive translation
4307     MODE_CapsLock  bool    // software CapsLock
4308     MODE_LowerLock bool    // force lower-case character lock
4309     MODE_ViInsert   int     // visible insert mode, should be like "I" icon in X Window
4310     MODE_ViTrace   bool    // output newline before translation
4311 }
4312 type IInput struct {
4313     lno      int
4314     lastlno  int
4315     pch     []int // input queue
4316     prompt   string
4317     line     string
4318     right    string
4319     inMode   bool
4320     pinJMode bool
4321     waitingMeta string // waiting meta character
4322     LastCmd   string
4323 }
4324 func (iin*IInput)getc(timeoutUs int)(int){
4325     ch1 := EOF
4326     ch2 := EOF
4327     ch3 := EOF
4328     if( 0 < len(iin.pch) ){ // deQ
4329         ch1 = iin.pch[0]
4330         iin.pch = iin.pch[1:]
4331     }else{
4332         ch1 = fgetcTimeout(stdin,timeoutUs);
4333     }
4334     if( ch1 == 033 ){ // escape sequence
4335         ch2 = fgetcTimeout(stdin,EscTimeout);
4336         if( ch2 == EV_TIMEOUT ){
4337             ch3 = fgetcTimeout(stdin,EscTimeout);
4338             if( ch3 == EV_TIMEOUT ){
4339                 iin.pch = append(iin.pch,ch2) // enQ
4340             }else{
4341                 switch( ch2 ){
4342                     default:
4343                         iin.pch = append(iin.pch,ch2) // enQ
4344                         iin.pch = append(iin.pch,ch3) // enQ
4345                     case '[':
4346                         switch( ch3 ){
4347                             case 'A': ch1 = GO_UP; // ^
4348                             case 'B': ch1 = GO_DOWN; // v
4349                             case 'C': ch1 = GO_RIGHT; // >
4350                             case 'D': ch1 = GO_LEFT; // <
4351                             case '3':
4352                                 ch4 := fgetcTimeout(stdin,EscTimeout);
4353                                 if( ch4 == '-' ){
4354                                     //fprintf(stderr,"%02X %02X %02X]\n",ch1,ch2,ch3,ch4);
4355                                     ch1 = DEL_RIGHT
4356                                 }
4357                         }
4358                     case '\\':
4359                         //ch4 := fgetcTimeout(stdin,EscTimeout);
4360                         //fprintf(stderr,"y%02X %02X %02X]\n",ch1,ch2,ch3,ch4);
4361                         switch( ch3 ){
4362                             case '-': ch1 = DEL_RIGHT
4363                         }
4364                 }
4365             }
4366         }
4367     }
4368 }
4369 return ch1
4370 }
4371 func (inn*IInput)clearline(){
4372     var i int
4373     fprintf(stderr,"\r");
4374     // should be ANSI ESC sequence

```

```

4375     for i = 0; i < TtyMaxCol; i++ { // to the max. position in this input action
4376         fputc(' ',os.Stderr);
4377     }
4378     fprintf(stderr,"\r");
4379 }
4380 func (iin*IInput)Redraw(){
4381     redraw(iin,iin.lno,iin.line,iin.right)
4382 }
4383 func redraw(iin *IInput,lno int,line string,right string){
4384     inMeta := false
4385     showMode := ""
4386     showMeta := "" // visible Meta mode on the cursor position
4387     showLino := fmt.Sprintf("%d! ",lno)
4388     InsertMark := "" // in visible insert mode
4389
4390     if MODE_VicMode {
4391     }else{
4392     if 0 < len(iin.right) {
4393         InsertMark = " "
4394     }
4395
4396     if( 0 < len(iin.waitingMeta) ){
4397         inMeta = true
4398         if iin.waitingMeta[0] != 033 {
4399             showMeta = iin.waitingMeta
4400         }
4401     }
4402     if( romkanmode ){
4403         //romkanmark = " *";
4404     }else{
4405         //romkanmark = "";
4406     }
4407     if MODE_ShowMode {
4408         romkan := "--"
4409         inmeta := "."
4410         inveri := ""
4411         if MODE_CapsLock {
4412             inmeta = "A"
4413         }
4414         if MODE_LowerLock {
4415             inmeta = "a"
4416         }
4417         if MODE_ViTrace {
4418             inveri = "v"
4419         }
4420         if MODE_VicMode {
4421             inveri = ":"
4422         }
4423         if romkanmode {
4424             romkan = "\343\201\202"
4425             if MODE_CapsLock {
4426                 inmeta = "R"
4427             }else{
4428                 inmeta = "r"
4429             }
4430         if inMeta {
4431             inmeta = "\\\"
4432         }
4433     }
4434     showMode = "["+romkan+inmeta+inveri+"]";
4435 }
4436 Pre := "\r" + showMode + showLino
4437 Output := ""
4438 Left := ""
4439 Right := ""
4440 if romkanmode {
4441     Left = convs(line)
4442     Right = InsertMark+convs(right)
4443 }else{
4444     Left = line
4445     Right = InsertMark+right
4446 }
4447 Output = Pre+Left
4448 if MODE_ViTrace {
4449     Output += iin.LastCmd
4450 }
4451 Output += showMeta+Right
4452 for len(Output) < TtyMaxCol { // to the max. position that may be dirty
4453     Output += " "
4454     // should be ANSI ESC sequence
4455     // not necessary just after newline
4456 }
4457 Output += Pre+Left+showMeta // to set the cursor to the current input position
4458 fprintf(stderr,"%s",Output)
4459
4460 if MODE_ViTrace {
4461     if 0 < len(iin.LastCmd) {
4462         iin.LastCmd = ""
4463         fprintf(stderr,"\r\n")
4464     }
4465 }
4466 // <a href="https://golang.org/pkg/unicode/utf8/">utf8</a>
4467 func delHeadChar(str string)(rline string,head string){
4468     _,cлен := utf8.DecodeRune([]byte(str))
4469     head = string(str[0:cлен])
4470     return str[cлен:],head
4471 }
4472 }
4473 func delTailChar(str string)(rline string, last string){
4474     var i = 0
4475     var clen = 0
4476     for {
4477         _,siz := utf8.DecodeRune([]byte(str)[i:])
4478         if siz <= 0 { break }
4479         clen = siz
4480         i += siz
4481     }
4482     last = str[len(str)-clen:]
4483     return str[0:len(str)-clen],last
4484 }
4485
4486 // 3> for output and history
4487 // 4> for keylog?
4488 // <a name="getline">Command Line Editor</a>
4489 func xgetline(iin int, prevline string, gsh*GshContext)(string){
4490     var iin IInput
4491     iin.lastlno = lno
4492     iin.lno = lno
4493
4494     CmdIndex = len(gsh.CommandHistory)
4495     if( isatty(0) == 0 ){
4496         if( sfgets(&iin.line,LINESIZE,stdin) == NULL ){
4497             iin.line = "exit\n";
4498         }else{
4499     }

```

```

4500     return iin.line
4501 }
4502 if( true ){
4503     //var pts string;
4504     //pts = ptsname(0);
4505     //pts = ttyname(0);
4506     //fprintf(stderr,"--pts[0] = %s\n",pts?pts:"?");
4507 }
4508 if( false ){
4509     fprintf(stderr,"! ");
4510     fflush(stderr);
4511     sfgets(&iin.line,LINESIZE,stdin);
4512     return iin.line
4513 }
4514 system("/bin/stty -echo -icanon");
4515 xline := iin.xgetline1(prevline,gsh)
4516 system("/bin/stty echo sane");
4517 return xline
4518 }
4519 func (in*IInput)Translate(cmdch int){
4520     romkanemode = !romkanemode;
4521     if MODE_ViTrace {
4522         fprintf(stderr,"%v\r\n",string(cmdch));
4523     }else
4524     if( cmdch == 'J' ){
4525         fprintf(stderr,"J\r\n");
4526         iin.inJMode = true
4527     }
4528     iin.Redraw();
4529     loadDefaultDic(cmdch);
4530     iin.Redraw();
4531 }
4532 func (in*IInput)Replace(cmdch int){
4533     iin.LastCmd = fmt.Sprintf("\\$v",string(cmdch))
4534     iin.Redraw();
4535     loadDefaultDic(cmdch);
4536     dst := convs(iin.line+iin.right);
4537     iin.line = dst
4538     iin.right = ""
4539     if( cmdch == 'I' ){
4540         fprintf(stderr,"I\r\n");
4541         iin.inJMode = true
4542     }
4543     iin.Redraw();
4544 }
4545 // aa 12 alal
4546 func isAlpha(ch rune)(bool){
4547     if 'a' <= ch && ch <= 'z' || 'A' <= ch && ch <= 'z' {
4548         return true
4549     }
4550     return false
4551 }
4552 func isAlnum(ch rune)(bool){
4553     if 'a' <= ch && ch <= 'z' || 'A' <= ch && ch <= 'z' {
4554         return true
4555     }
4556     if '0' <= ch && ch <= '9' {
4557         return true
4558     }
4559     return false
4560 }
4561 // 0.2.8 2020-0901 created
4562 // <a href="https://golang.org/pkg/unicode/utf8/#DecodeRuneInString">DecodeRuneInString</a>
4563 func (in*IInput)GotoTOPW(){
4564     str := iin.line
4565     i := len(str)
4566     if i <= 0 {
4567         return
4568     }
4569     //i0 := i
4570     i -= 1
4571     lastSize := 0
4572     var lastRune rune
4573     var found = -1
4574     for 0 < i { // skip preamble spaces
4575         lastRune,lastSize = utf8.DecodeRuneInString(str[i:])
4576         if !isAlnum(lastRune) { // character, type, or string to be searched
4577             i -= lastSize
4578             continue
4579         }
4580     }
4581     break
4582 }
4583 for 0 < i {
4584     lastRune,lastSize = utf8.DecodeRuneInString(str[i:])
4585     if lastSize <= 0 { continue } // not the character top
4586     if !isAlnum(lastRune) { // character, type, or string to be searched
4587         found = i
4588         break
4589     }
4590     i -= lastSize
4591 }
4592 if found < 0 && i == 0 {
4593     found = 0
4594 }
4595 if 0 <= found {
4596     if isAlnum(lastRune) { // or non-kana character
4597     }else{ // when positioning to the top o the word
4598         i += lastSize
4599     }
4600     iin.right = str[i:] + iin.right
4601     if 0 < i {
4602         iin.line = str[0:i]
4603     }else{
4604         iin.line = ""
4605     }
4606 }
4607 //fmt.Printf("\n(%d,%d,%d)[%s]\n",i0,i,found,iin.line,iin.right)
4608 //fmt.Println("") // set debug messae at the end of line
4609 }
4610 // 0.2.8 2020-0901 created
4611 func (in*IInput)GotoENDW(){
4612     str := iin.right
4613     if len(str) <= 0 {
4614         return
4615     }
4616     lastSize := 0
4617     var lastRune rune
4618     var lastW = 0
4619     i := 0
4620     inWord := false
4621
4622     lastRune,lastSize = utf8.DecodeRuneInString(str[0:])
4623     if isAlnum(lastRune) {
4624         r,g,i := utf8.DecodeRuneInString(str[lastSize-1])
4625         if r >= 'a' && r <= 'z' && g >= 'A' && g <= 'Z' {
4626             inWord = true
4627         }
4628     }
4629     for i < len(str) {
4630         lastRune,lastSize = utf8.DecodeRuneInString(str[i:])
4631         if !isAlnum(lastRune) {
4632             if inWord {
4633                 iin.line = str[0:i]
4634             }
4635             break
4636         }
4637     }
4638     if inWord {
4639         iin.line = str[0:i]
4640     }
4641 }

```

```

4625     if 0 < z && isAlnum(r) {
4626         inWord = true
4627     }
4628 }
4629 for i < len(str) {
4630     lastRune,lastSize = utf8.DecodeRuneInString(str[i:])
4631     if lastSize <= 0 { break } // broken data?
4632     if !isAlnum(lastRune) { // character, type, or string to be searched
4633         break
4634     }
4635     lastW = i // the last alnum if in alnum word
4636     i += lastSize
4637 }
4638 if inWord {
4639     goto DISP
4640 }
4641 for i < len(str) {
4642     lastRune,lastSize = utf8.DecodeRuneInString(str[i:])
4643     if lastSize <= 0 { break } // broken data?
4644     if isAlnum(lastRune) { // character, type, or string to be searched
4645         break
4646     }
4647     i += lastSize
4648 }
4649 for i < len(str) {
4650     lastRune,lastSize = utf8.DecodeRuneInString(str[i:])
4651     if lastSize <= 0 { break } // broken data?
4652     if !isAlnum(lastRune) { // character, type, or string to be searched
4653         break
4654     }
4655     lastW = i
4656     i += lastSize
4657 }
4658 DISP:
4659     if 0 < lastW {
4660         iin.line = iin.line + str[0:lastW]
4661         iin.right = str[lastW:]
4662     }
4663 //fmt.Printf("\n%d[%s]\n",i,iin.line,iin.right)
4664 //fmt.Printf("") // set debug messae at the end of line
4665 }
4666 // 0.2.8 2020-0901 created
4667 func (iin*IInput)GotoNEXTW(){
4668     str := iin.right
4669     if len(str) <= 0 {
4670         return
4671     }
4672     lastSize := 0
4673     var lastRune rune
4674     var found = -1
4675     i := 1
4676     for i < len(str) {
4677         lastRune,lastSize = utf8.DecodeRuneInString(str[i:])
4678         if lastSize <= 0 { break } // broken data?
4679         if !isAlnum(lastRune) { // character, type, or string to be searched
4680             found = i
4681             break
4682         }
4683         i += lastSize
4684     }
4685     if 0 < found {
4686         if isAlnum(lastRune) { // or non-kana character
4687             }else{ // when positioning to the top o the word
4688                 found += lastSize
4689             }
4690         iin.line = iin.line + str[0:found]
4691         if 0 < found {
4692             iin.right = str[found:]
4693         }else{
4694             iin.right = ""
4695         }
4696     }
4697 //fmt.Printf("\n%d[%s]\n",i,iin.line,iin.right)
4698 //fmt.Printf("") // set debug messae at the end of line
4699 }
4700 // 0.2.8 2020-0902 created
4701 func (iin*IInput)GotoPAIRCH(){
4702     str := iin.right
4703     if len(str) <= 0 {
4704         return
4705     }
4706     lastRune,lastSize := utf8.DecodeRuneInString(str[0:])
4707     if lastSize <= 0 {
4708         return
4709     }
4710     forw := false
4711     back := false
4712     pair := ""
4713     switch string(lastRune){
4714         case "(": pair = ")"; forw = true
4715         case ")": pair = "("; back = true
4716         case "{": pair = "}"; forw = true
4717         case ")": pair = "("; back = true
4718         case "[": pair = "]"; forw = true
4719         case "]": pair = "["; back = true
4720         case "<": pair = ">"; forw = true
4721         case ">": pair = "<"; back = true
4722         case "\"": pair = "\""; // context depednet, can be f" or back-double quote
4723         case "'": pair = "'"; // context depednet, can be f' or back-quote
4724         // case Japanese Kakkos
4725     }
4726     if forw {
4727         iin.SearchForward(pair)
4728     }
4729     if back {
4730         iin.SearchBackward(pair)
4731     }
4732 }
4733 // 0.2.8 2020-0902 created
4734 func (iin*IInput)SearchForward(pat string)(bool){
4735     right := iin.right
4736     found := -1
4737     i := 0
4738     if strBegins(right,pat) {
4739         z := utf8.DecodeRuneInString(right[i:])
4740         if 0 < z {
4741             i += z
4742         }
4743     }
4744     for i < len(right) {
4745         if strBegins(right[i:],pat) {
4746             found = i
4747             break
4748         }
4749     }
4750     z := utf8.DecodeRuneInString(right[i:])

```

```

4750     if z <= 0 { break }
4751     i += z
4752   }
4753   if 0 <= found {
4754     iin.line = iin.line + right[0:found]
4755     iin.right = iin.right[found:]
4756     return true
4757   }else{
4758     return false
4759   }
4760 }
4761 // 0.2.8 2020-0902 created
4762 func (iin*IInput)SearchBackward(pat string)(bool){
4763   line := iin.line
4764   found := -1
4765   i := len(line)-1
4766   for i = i; 0 <= i; i-- {
4767     z := utf8.DecodeRuneInString(line[i:])
4768     if z <= 0 {
4769       continue
4770     }
4771     //fprintf(stderr,"-- %v %v\n",pat,line[i:])
4772     if strBegins(line[i:],pat) {
4773       found = i
4774       break
4775     }
4776   }
4777   //fprintf(stderr,"--%d\n",found)
4778   if 0 <= found {
4779     iin.right = line[found:] + iin.right
4780     iin.line = line[0:found]
4781     return true
4782   }else{
4783     return false
4784   }
4785 }
4786 // 0.2.8 2020-0902 created
4787 // search from top, end, or current position
4788 func (gsh*GshContext)SearchHistory(pat string, forw bool)(bool,string){
4789   if forw {
4790     for v := range gsh.CommandHistory {
4791       if 0 <= strings.Index(v.CmdLine,pat) {
4792         //fprintf(stderr,"\n--De-- found !%v [%v]\n",i,pat,v.CmdLine)
4793         return true,v.CmdLine
4794       }
4795     }
4796   }else{
4797     hlen := len(gsh.CommandHistory)
4798     for i := hlen-1; 0 < i ; i-- {
4799       v := gsh.CommandHistory[i]
4800       if 0 <= strings.Index(v.CmdLine,pat) {
4801         //fprintf(stderr,"\n--De-- found !%v [%v]\n",i,pat,v.CmdLine)
4802         return true,v.CmdLine
4803       }
4804     }
4805   }
4806   //fprintf(stderr,"n--De-- not-found(%v)\n",pat)
4807   return false,"(Not Found in History)"
4808 }
4809 // 0.2.8 2020-0902 created
4810 func (iin*IInput)GotoFORWSTR(pat string,gsh*GshContext){
4811   found := false
4812   if 0 < len(iin.right) {
4813     found = iin.SearchForward(pat)
4814   }
4815   if !found {
4816     found,line := gsh.SearchHistory(pat,true)
4817     if found {
4818       iin.line = line
4819       iin.right = ""
4820     }
4821   }
4822 }
4823 func (iin*IInput)GotoBACKSTR(pat string, gsh*GshContext){
4824   found := false
4825   if 0 < len(iin.line) {
4826     found = iin.SearchBackward(pat)
4827   }
4828   if !found {
4829     found,line := gsh.SearchHistory(pat,false)
4830     if found {
4831       iin.line = line
4832       iin.right = ""
4833     }
4834   }
4835 }
4836 func (iin*IInput)getstring1(prompt string)(string){ // should be editable
4837   iin.clearline();
4838   fprintf(stderr,"r%v",prompt)
4839   str := ""
4840   for {
4841     ch := iin.Getc(10*1000*1000)
4842     if ch == '\n' || ch == '\r' {
4843       break
4844     }
4845     sch := string(ch)
4846     str += sch
4847     fprintf(stderr,"%s",sch)
4848   }
4849   return str
4850 }
4851
4852 // search pattern must be an array and selectable with ^N/^P
4853 var SearchPat = ""
4854 var SearchForw = true
4855
4856 func (iin*IInput)xgetline1(prevline string, gsh*GshContext)(string{
4857   var ch int;
4858
4859   MODE_ShowMode = false
4860   MODE_VicMode = false
4861   iin.Redraw();
4862   first := true
4863
4864   for cix := 0; ; cix++ {
4865     iin.pinJmode = iin.inJmode
4866     iin.inJmode = false
4867
4868     ch = iin.Getc(1000*1000)
4869
4870     if ch != EV_TIMEOUT && first {
4871       first = false
4872       mode := 0
4873       if romkanmode {
4874         mode = 1

```

```

4875
4876     }
4877     now := time.Now()
4878     Events = append(Events, Event{now, EV_MODE, int64(mode), CmdIndex})
4879 }
4880 if ch == 033 {
4881     MODE_ShowMode = true
4882     MODE_VicMode = !MODE_VicMode
4883     iin.Redraw();
4884     continue
4885 }
4886 if MODE_VicMode {
4887     switch ch {
4888         case '0': ch = GO_TOPL
4889         case 'S': ch = GO_ENDL
4890         case 'B': ch = GO_TOPW
4891         case 'E': ch = GO_ENDW
4892         case 'W': ch = GO_NEXTW
4893         case 'S': ch = GO_PAIRCH
4894
4895         case 'j': ch = GO_DOWN
4896         case 'k': ch = GO_UP
4897         case 'h': ch = GO_LEFT
4898         case 'l': ch = GO_RIGHT
4899         case 'x': ch = DEL_RIGHT
4900         case 'a': MODE_VicMode = !MODE_VicMode
4901             ch = GO_RIGHT
4902         case 'i': MODE_VicMode = !MODE_VicMode
4903             iin.Redraw();
4904             continue
4905         case '-':
4906             right,head := delHeadChar(iin.right)
4907             if len([]byte(head)) == 1 {
4908                 ch = int(head[0])
4909                 if( 'a' <= ch && ch <= 'z' ){
4910                     ch = ch + 'A'-'a'
4911                 }else{
4912                     if( 'A' <= ch && ch <= 'Z' ){
4913                         ch = ch + 'a'-'A'
4914                     }
4915                 iin.right = string(ch) + right
4916             }
4917             iin.Redraw();
4918             continue
4919         case 'f': // GO_FORWCH
4920             iin.Redraw();
4921             ch = iin.Getc(3*1000*1000)
4922             if ch == EV_TIMEOUT {
4923                 iin.Redraw();
4924                 continue
4925             }
4926             SearchPat = string(ch)
4927             SearchForw = true
4928             iin.GotoFORWSTR(SearchPat,gsh)
4929             iin.Redraw();
4930             continue
4931         case '/':
4932             SearchPat = iin.getstring1("// should be editable")
4933             SearchForw = true
4934             iin.GotoFORWSTR(SearchPat,gsh)
4935             iin.Redraw();
4936             continue
4937         case '?':
4938             SearchPat = iin.getstring1("// should be editable")
4939             SearchForw = false
4940             iin.GotoBACKSTR(SearchPat,gsh)
4941             iin.Redraw();
4942             continue
4943         case 'n':
4944             if SearchForw {
4945                 iin.GotoFORWSTR(SearchPat,gsh)
4946             }else{
4947                 iin.GotoBACKSTR(SearchPat,gsh)
4948             }
4949             iin.Redraw();
4950             continue
4951         case 'N':
4952             if !SearchForw {
4953                 iin.GotoFORWSTR(SearchPat,gsh)
4954             }else{
4955                 iin.GotoBACKSTR(SearchPat,gsh)
4956             }
4957             iin.Redraw();
4958             continue
4959     }
4960     switch ch {
4961         case GO_TOPW:
4962             iin.GotoTOPW()
4963             iin.Redraw();
4964             continue
4965         case GO_ENDW:
4966             iin.GotoENDW()
4967             iin.Redraw();
4968             continue
4969         case GO_NEXTW:
4970             // to next space then
4971             iin.GotoNEXTW()
4972             iin.Redraw();
4973             continue
4974         case GO_PAIRCH:
4975             iin.GotoPAIRCH()
4976             iin.Redraw();
4977             continue
4978     }
4979
4980 //fprintf(stderr,"A[%02X]\n",ch);
4981 if( ch == '\\' || ch == 033 ){
4982     MODE_ShowMode = true
4983     metach := ch
4984     iin.waitingMeta = string(ch)
4985     iin.Redraw();
4986     // set cursor //fprintf(stderr,"???\b\b\b")
4987     ch = fgetcTimeout(stdin,2000*1000)
4988     // reset cursor
4989     iin.waitingMeta = ""
4990
4991     cmdch := ch
4992     if( ch == EV_TIMEOUT ){
4993         if metach == 033 {
4994             continue
4995         }
4996         ch = metach
4997     }else
4998 /* */
4999 if( ch == 'm' || ch == 'M' ){

```

```

5000     mch := fgetcTimeout(stdin,1000*1000)
5001     if mch == 'r' {
5002         romkanmode = true
5003     }else{
5004         romkanmode = false
5005     }
5006     continue
5007 }else
5008 */
5009 if( ch == 'k' || ch == 'K' ){
5010     MODE_Recursive = !MODE_Recursive
5011     iin.Translate(cmdch);
5012     continue
5013 }else
5014 if( ch == 'j' || ch == 'J' ){
5015     iin.Translate(cmdch);
5016     continue
5017 }else
5018 if( ch == 'i' || ch == 'I' ){
5019     iin.Replace(cmdch);
5020     continue
5021 }else
5022 if( ch == 'l' || ch == 'L' ){
5023     MODE_LowerLock = !MODE_LowerLock
5024     MODE_CapsLock = false
5025     if MODE_ViTrace {
5026         fprintf(stderr,"%v\r\n",string(cmdch));
5027     }
5028     iin.Redraw();
5029     continue
5030 }else
5031 if( ch == 'u' || ch == 'U' ){
5032     MODE_CapsLock = !MODE_CapsLock
5033     MODE_LowerLock = false
5034     if MODE_ViTrace {
5035         fprintf(stderr,"%v\r\n",string(cmdch));
5036     }
5037     iin.Redraw();
5038     continue
5039 }else
5040 if( ch == 'v' || ch == 'V' ){
5041     MODE_ViTrace = !MODE_ViTrace
5042     if MODE_ViTrace {
5043         fprintf(stderr,"%v\r\n",string(cmdch));
5044     }
5045     iin.Redraw();
5046     continue
5047 }else
5048 if( ch == 'c' || ch == 'C' ){
5049     if 0 < len(iin.line) {
5050         xline,tail := delTailChar(iin.line)
5051         if len([]byte(tail)) == 1 {
5052             ch = int(tail[0])
5053             if( 'a' <= ch && ch <= 'z' ){
5054                 ch = ch + 'A'-'a'
5055             }else
5056             if( 'A' <= ch && ch <= 'Z' ){
5057                 ch = ch + 'a'-'A'
5058             }
5059             iin.line = xline + string(ch)
5060         }
5061         if MODE_ViTrace {
5062             fprintf(stderr,"%v\r\n",string(cmdch));
5063         }
5064         iin.Redraw();
5065         continue
5066     }else{
5067         iin.pch = append(iin.pch,ch) // push
5068         ch = '\\'
5069     }
5070 }
5071 switch( ch ){
5072     case 'P'-0x40: ch = GO_UP
5073     case 'N'-0x40: ch = GO_DOWN
5074     case 'B'-0x40: ch = GO_LEFT
5075     case 'F'-0x40: ch = GO_RIGHT
5076 }
//fprintf(stderr,"B[%02X]\n",ch);
5077 switch( ch ){
5078     case 0:
5079         continue;
5080
5081     case '\t':
5082         iin.Replace('j');
5083         continue;
5084     case 'X'-0x40:
5085         iin.Replace('j');
5086         continue;
5087
5088     case EV_TIMEOUT:
5089         iin.Redraw();
5090         if iin.pinJMode {
5091             fprintf(stderr,"\\J\r\n")
5092             iin.inJMode = true
5093         }
5094         continue;
5095     case GO_UP:
5096         if iin.lno == 1 {
5097             continue;
5098         }
5099         cmd,ok := gsh.cmdStringInHistory(iin.lno-1)
5100         if ok {
5101             iin.line = cmd
5102             iin.right = ""
5103             iin.lno = iin.lno - 1
5104         }
5105         iin.Redraw();
5106         continue;
5107     case GO_DOWN:
5108         cmd,ok := gsh.cmdStringInHistory(iin.lno+1)
5109         if ok {
5110             iin.line = cmd
5111             iin.right = ""
5112             iin.lno = iin.lno + 1
5113         }else{
5114             iin.line = ""
5115             iin.right = ""
5116             if iin.lno == iin.lastlno-1 {
5117                 iin.lno = iin.lno + 1
5118             }
5119         }
5120         iin.Redraw();
5121         continue;
5122     case GO_LEFT:
5123

```

```

5125     if( 0 < len(iin.line) ){
5126         xline,tail := delTailChar(iin.line)
5127         iin.line = xline
5128         iin.right = tail + iin.right
5129     }
5130     iin.Redraw();
5131     continue;
5132   case GO_RIGHT:
5133     if( 0 < len(iin.right) && iin.right[0] != 0 ){
5134       xright,head := delHeadChar(iin.right)
5135       iin.right = xright
5136       iin.line += head
5137     }
5138     iin.Redraw();
5139     continue;
5140   case EOF:
5141     goto EXIT;
5142   case 'R'-0x40: // replace
5143     dst := convs(iin.line+iin.right);
5144     iin.line = dst
5145     iin.right = ""
5146     iin.Redraw();
5147     continue;
5148   case 'T'-0x40: // just show the result
5149     readDic();
5150     romkanmode = !romkanmode;
5151     iin.Redraw();
5152     continue;
5153   case 'L'-0x40:
5154     iin.Redraw();
5155     continue;
5156   case 'K'-0x40:
5157     iin.right = ""
5158     iin.Redraw();
5159     continue;
5160   case 'E'-0x40:
5161     iin.line += iin.right
5162     iin.right = ""
5163     iin.Redraw();
5164     continue;
5165   case 'A'-0x40:
5166     iin.right = iin.line + iin.right
5167     iin.line = ""
5168     iin.Redraw();
5169     continue;
5170   case 'U'-0x40:
5171     iin.line = ""
5172     iin.right = ""
5173     iin.clearline();
5174     iin.Redraw();
5175     continue;
5176   case DEL_RIGHT:
5177     if( 0 < len(iin.right) ){
5178       iin.right,_ = delHeadChar(iin.right)
5179       iin.Redraw();
5180     }
5181     continue;
5182   case 0x7F: // BS? not DEL
5183     if( 0 < len(iin.line) ){
5184       iin.line,_ = delTailChar(iin.line)
5185       iin.Redraw();
5186     }
5187     /*
5188     else
5189     if( 0 < len(iin.right) ){
5190       iin.right,_ = delHeadChar(iin.right)
5191       iin.Redraw();
5192     }
5193     */
5194     continue;
5195   case 'H'-0x40:
5196     if( 0 < len(iin.line) ){
5197       iin.line,_ = delTailChar(iin.line)
5198       iin.Redraw();
5199     }
5200     continue;
5201   if( ch == '\n' || ch == '\r' ){
5202     iin.line += iin.right;
5203     iin.right = "";
5204     iin.Redraw();
5205     fputc(ch,stderr);
5206     break;
5207   }
5208   if MODE_CapsLock {
5209     if 'a' <= ch && ch <= 'z' {
5210       ch = ch+'A'-'a'
5211     }
5212   }
5213   if MODE_LowerLock {
5214     if 'A' <= ch && ch <= 'Z' {
5215       ch = ch+'a'-'A'
5216     }
5217   }
5218   iin.line += string(ch);
5219   iin.Redraw();
5220 }
5221 }
5222 EXIT:
5223   return iin.line + iin.right;
5224 }
5225
5226 func getline_main(){
5227   line := xgetline(0,"",nil)
5228   fprintf(stderr,"%s\n",line);
5229   /*
5230   dp = strpbrk(line,"\r\n");
5231   if( dp != NULL ){
5232     *dp = 0;
5233   }
5234
5235   if( 0 ){
5236     fprintf(stderr,"%n(%d)\n",int(strlen(line)));
5237   }
5238   if( lseek(3,0,0) == 0 ){
5239     if( romkanmode ){
5240       var buf [8*1024]byte;
5241       convs(line,buf);
5242       strcpy(line,buf);
5243     }
5244     write(3,line,strlen(line));
5245     ftruncate(3,lseek(3,0,SEEK_CUR));
5246     //fprintf(stderr,"outsize=%d\n", (int)lseek(3,0,SEEK_END));
5247     lseek(3,0,SEEK_SET);
5248     close(3);
5249   }else{

```

```

5250     fprintf(stderr,"\\r\\ngotline: ");
5251     trans(line);
5252     //printf("%s\\n",line);
5253     printf("\\n");
5254 }
5255 */
5256 }
5257 //== end ===== getline
5258
5259 //
5260 // $USERHOME/.gsh/
5261 //   gsh-rc.txt, or gsh-configure.txt
5262 //   gsh-history.txt
5263 //   gsh-aliases.txt // should be conditional?
5264 //
5265 func (gshCtx *GshContext)gshSetupHomedir()(bool) {
5266     homedir,found := userHomeDir()
5267     if !found {
5268         fmt.Printf("--E-- You have no UserHomeDir\\n")
5269         return true
5270     }
5271     gshhome := homedir + "/" + GSH_HOME
5272     _,err2 := os.Stat(gshhome)
5273     if err2 != nil {
5274         err3 := os.Mkdir(gshhome,0700)
5275         if err3 != nil {
5276             fmt.Printf("--E-- Could not Create %s (%s)\\n",
5277                   gshhome,err3)
5278             return true
5279         }
5280         fmt.Printf("--I-- Created %s\\n",gshhome)
5281     }
5282     gshCtx.GshHomeDir = gshhome
5283     return false
5284 }
5285 func setupGshContext()(GshContext,bool){
5286     gshPA := syscall.ProcAttr {
5287         "", // the starting directory
5288         os.Environ(), // environ[]
5289         []uintptr{os.Stdin.Fd(),os.Stdout.Fd(),os.Stderr.Fd()},
5290         nil, // OS specific
5291     }
5292     cwd, _ := os.Getwd()
5293     gshctx := GshContext {
5294         cwd, // StartDir
5295         "", // GetLine
5296         [1]GChdirHistory { { cwd,time.Now(),0 } }, // ChdirHistory
5297         gshPA,
5298         []GCommandHistory{}, //something for invocation?
5299         GCommandHistory{}, // CmdCurrent
5300         false,
5301         []int{},
5302         syscall.Rusage{},
5303         "", // GshHomeDir
5304         Ttyid(),
5305         false,
5306         false,
5307         []PluginInfo{},
5308         []string{},
5309         " ",
5310         "v",
5311         ValueStack{},
5312         GServer{"",""}, // LastServer
5313         "", // RSERV
5314         cwd, // RWD
5315         CheckSum{},
5316     }
5317     err := gshctx.gshSetupHomedir()
5318     return gshctx, err
5319 }
5320 func (gsh*GshContext)gshellh(gline string)(bool){
5321     ghist := gsh.CmdCurrent
5322     ghist.WorkDir,_ = os.Getwd()
5323     ghist.WorkDirX = len(gsh.ChdirHistory)-1
5324     //fmt.Printf("--D--ChdirHistory(@%d)\\n",len(gsh.ChdirHistory))
5325     ghist.StartAt = time.Now()
5326     usageev1 := Getrusagev()
5327     gsh.CmdCurrent.FoundFile = []string{}
5328     fin := gsh.tgshell(gline)
5329     usageev2 := Getrusagev()
5330     ghist.Rusagev = RusageSubv(usageev2,usageev1)
5331     ghist.EndAt = time.Now()
5332     ghist.CmdLine = gline
5333     ghist.FoundFile = gsh.CmdCurrent.FoundFile
5334
5335     /* record it but not show in list by default
5336     if len(gline) == 0 {
5337         continue
5338     }
5339     if gline == "hi" || gline == "history" { // don't record it
5340         continue
5341     }
5342     */
5343     gsh.CommandHistory = append(gsh.CommandHistory, ghist)
5344     return fin
5345 }
5346 // <a name="main">Main loop</a>
5347 func script(gshCtxGiven *GshContext) (_ GshContext) {
5348     gshctxBuf,err0 := setupGshContext()
5349     if err0 {
5350         return gshctxBuf;
5351     }
5352     gshctx := &gshctxBuf
5353
5354     //fmt.Printf("--I-- GSH_HOME=%s\\n",gshctx.GshHomeDir)
5355     //resmap()
5356
5357 /*
5358 if false {
5359     gsh_getlinev, with_exgetline :=
5360         which("PATH",[]string{"which","gsh-getline","-s"})
5361     if with_exgetline {
5362         gsh_getlinev[0] = toFullPath(gsh_getlinev[0])
5363         gshctx.GetLine = toFullPath(gsh_getlinev[0])
5364     }else{
5365         fmt.Printf("--W-- No gsh-getline found. Using internal getline.\\n");
5366     }
5367 }
5368 */
5369
5370 ghist0 := gshctx.CmdCurrent // something special, or gshrc script, or permanent history
5371 gshctx.CommandHistory = append(gshctx.CommandHistory,ghist0)
5372
5373 prevline := ""
5374 skipping := false

```

```

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



```

5625   width:320px; height:20px;
5626   color:#fff; background-color:rgba(0,0,0,0.5);
5627   color:#fff; font-size:12px;
5628 }
5629 #GPos{
5630   z-index:10;
5631   position:fixed; top:0px; left:0px;
5632   opacity:1.0;
5633   width:320px; height:30px;
5634   color:#fff; background-color:rgba(0,0,0,0.4);
5635   color:#fff; font-size:12px;
5636 }
5637 #GMenu{
5638   z-index:20;
5639   position:fixed; top:250px; left:0px;
5640   opacity:1.0;
5641   width:100px; height:170px;
5642   color:#fff; background-image:url(GShellInsideIcon);
5643   color:#fff; font-size:16px; font-family:Georgia;
5644   background-repeat:no-repeat;
5645 }
5646 #GStat{
5647   z-index:9;
5648   xopacity:0.0;
5649   position:fixed; top:20px; left:0px;
5650   width:320px; height:90px;
5651   color:#fff; background-color:rgba(0,0,0,0.4);
5652   font-size:20px; font-family:Georgia;
5653 }
5654 #GLog{
5655   z-index:10;
5656   position:fixed; top:50px; left:0px;
5657   opacity:1.0;
5658   width:320px; height:60px;
5659   color:#fff; background-color:rgba(0,0,0,0.3);
5660   font-size:12px;
5661 }
5662 #GshGrid {
5663   z-index:11;
5664   xopacity:0.0;
5665   position:fixed; top:0px; left:0px;
5666   width:320px; height:30px;
5667   color:#f9f; font-size:16px;
5668 }
5669 xbody {display:none;}
5670 .gsh-link{color:green;}
5671 #gsh {border-width:1px;margin:0;padding:0;}
5672 #gsh {font-family:monospace,Courier New;color:#ddf;font-size:8px;}
5673 #gsh header{height:100px;}
5674 #xgsh header{height:100px;background-image:url(GShell-Logo00.png);}
5675 #GshMenu{font-size:14pt;color:#f88;}
5676 #gsh-footer{height:100px;background-size:80px;background-repeat:no-repeat;}
5677 #gsh-note{color:#000;font-size:10pt;}
5678 #gsh h2{color:#24a;font-family:Georgia;font-size:18pt;}
5679 #gsh h3{color:#24a;font-family:Georgia;font-size:16pt;}
5680 #gsh details{color:#888;background-color:#fff;font-family:monospace;}
5681 #gsh summary{font-size:16pt;color:#fff;background-color:#0af;height:30px;}
5682 #gsh pre{font-size:1lpt;color:#223;background-color:#faffff;}
5683 #gsh a{color:#24a;}
5684 #gsh a[name]{color:#24a;font-size:16pt;}
5685 #gsh .gsh-src{white-space:pre;font-family:monospace,Courier New;font-size:1lpt;}
5686 #gsh .gsh-src{background-color:#faffff;color:#223;}
5687 #gsh-src-src{spellcheck:false}
5688 #src-frame-textarea{white-space:pre;font-family:monospace,Courier New;font-size:1lpt;}
5689 #src-frame-textarea{background-color:#faffff;color:#223;}
5690 .gsh-code {white-space:pre;font-family:monospace !important;}
5691 .gsh-code {color:#088;font-size:1lpt; background-color:#eef;}
5692 .gsh-golang-data {display:none;}
5693 #gsh-WinId {color:#000;font-size:14pt;}
5694
5695 .gsh-document {font-size:1lpt;background-color:#fff;font-family:Georgia;}
5696 .gsh-document {color:#000;background-color:#fff !important;}
5697 .gsh-document > h2{color:#000;background-color:#fff !important;}
5698 .gsh-document details{color:#000;background-color:#fff;font-family:Georgia;}
5699 .gsh-document p{max-width:550pt;color:#000;background-color:#fff;font-family:Georgia;}
5700 .gsh-document address{width:500pt;color:#000;background-color:#fff;font-family:Georgia;}
5701
5702 @media print {
5703   #gsh pre{font-size:1lpt !important;}
5704 }
5705 </style>
5706
5707 <!--
5708 // Logo image should be drawn by JavaScript from a meta-font.
5709 // CSS seems not follow line-splitted URL
5710 -->
5711 <script id="gsh-data">
5712 //GShellLogo-QR-ITS-more.jp.png"
5713 GShellLogo="data:image/png;base64,\
5714 iVBORw0KGgoAAAANSUhEUgAAQAAEAAAB/CAYAAADvs3f4AAAAAXNSR0IArs4c6QAAAHHlWe1m\
5715 TU0AkGAAAAGABAfAAUAAAABAAAAPgfbAAUAAAABAAAARGeoAAMAAAABAAIAIdpAAQAAAAB\
5716 AAAATgAAAAAAABIAAAAQAAAEgAAAABAAQgAQADAAAAQgABAACgAgAEAAAAAQgAgAAE\ \
5717 AAAAQWAHAAAAYAYQbAAUAAA1wsFlzAAALEWAAcMBAJqCgAAAAR3RJEFEUAHtnQuURFN2\ \
5718 x++t7ukz31cg0o/jY6osb8wgMzAvn7u4G+biSTR7YnQxdQFCGj2anWd12Ms1rkeuapnco\ \
5719 4iuJx7jriyZ50D0gmP2vIBElqggCoIMMA+mu+v/u/ ZMD9U1da6a2aUbv91Krg3vvdx6/d\ \
5720 fnXvdx8tBA8SIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\ \
5721 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\ \
5722 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\ \
5723 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIIFD14A8dLP2\ \
5724 2exs9H9+ftSKsdxhs1c2qgDE7yusS+1qaal1fnY5ys0kMwEptdK4Mqrz5UeEx1LysaxU15\ \
5725 npDiLKEZEZC1FIRM53JSUaq9Sccct61+2KK3Stw0Ny5reEGKJ7Qw7m0vKec2Tqo1zwo1jhFS\ \
5726 jboVHCSttMRb3USXEJhFu7dSdmFB2+xU4WWFVbbpMeZU1AE/hcKoGab66eKG01Nykh5PC\ \
5727 Hx2VVBKoRqkh3qekiy1da0fON560kd16w5BwommQ0lypzi0n9DlmXpFK/60p2/Piyof\ \
5728 N8mfNt+/JWNNGnjw9KqOToLVGScf2p2rilln3j10vK7ys0wVmzEuvPfflRKYdfOak2LRSB0q\ \
5729 zrwCoOG6gEhvgrAcj/dktj3g7dXXH4gKN6ARS0zpYzergs6Ra0zDqfk79SKTRXHu/e+9FN\ \
5730 L66as8pU/PN1pN1TLQJkSc73dPXR20ur7i1wpcC8QhbNnCyU1lrryOTqvYF5JfvqlB7jx\ \
5731 +cNHjBj5g9Tq89Ua640d40h20qt8TAfPeFuOU+wIC-KnyhK5FGEV0WGGAEBX83eXMoLY\ \
5732 rikbd9gHEP52V9gQ1489P6E6KJyYFbbQbnzlJg42fiesndHCwvUoeiVQob/5C9FY9D1UueOH\ \
5733 +zghUh9nSgQmr0uWgukr1RpjBD4Y6u0cQD5T0UW63z2d3Mhesy14V9isbdKyxbGH1CpFR\ \
5734 U36taACF79VF58NBEDHT0MBA74En+eWrWz+Lz+Qtw60AdB7QJujps/O7cOoBNBCeMU2\ \
5735 ttCu/co28LpvKE1TPFV8juRasEhbHvxar1guoeBPfyD04+oFeDby814t29xExSFAMOC\ \
5736 bgGpov01zgGGw4jf392xnHdc+Mwf3J7jfntZ2y1YJBXNt5KIKyck1sxXBld1d6Bmcenv\ \
5737 aJoyv/BaMcEvqEP46/2lnJjt9j17VL5Jz15Mtavp1QG1NHw5pDqXyNTQ1Z228nGcmG2ZV\ \
5738 qooFjsdyvVOA2zdfayid6fJ35CS4jXZK9h1r7e27m6p3t8LJpkf1cjpV1Htk/DJFU4jw1\ \
5739 l1mhxm5IR9fzzgRkX4w/C+QOSPE+krbIyrN3qEPtNAhsHalDs2xh505NcoPvEpgqcbm/8e\ \
5740 7/zdahPttag/mlkJ77UOVGxybTdx/Ex/PTfa/i7x7Ku+Cso1CxUwrohuxP16EV9H+ccVg1\ \
5741 pd/CFU42AK2IUP1vTK1L/sjyv5PVHqrT28NzvfUzvdwODG9yGoopuhmNLnfCt48YHL2qH\ \
5742 f/kBpxVu/43Q9xtg6tYcv1LXDC3fmWDOn9nbf21e7wKE1bOK65icBu0Eqhd3iaW82wdKPUw\ \
5743 hrauc6zCwdkcjU2K8XUMae71zUgwC2nbi6vn1j9/P7ew+iGmAogP+N13iLsf8dn9ipA\ \
5744 WWN4rPy9jxu0eDL/HXZNgzTsves1d2vsWH19mu5rvVzX9fs4v/LfmqdEIpHDGlFM2uCW\ \
5745 gJY1z2wENPaZ3fcEcid7/76/rbjKy1d5pB25W14wQ07ut1kPbeCOpwK+jQsOp9GHN0AZuw1\ \
5746 b1WwbffLcRF04qazRd7176/rbjKy1d5pB25W14wQ07ut1kPbeCOpwK+jQsOp9GHN0AZuw1\ \
5747 io2uywDhQ9zBr2xoDRqVQF15Qxh60WvJRKAAN46pvT+RxAJVLjW7VY9/+CeUBMK168/rPQn\ \
5748 mcUfk2zaldrF/N/18gA5ivc3dIKhsyvzuCYSVG/KHcwHFWDRKAMCcD8EXK++HF12A9bt2d172\ \
5749 2qNzOvzCDYmfEtNy7QogxDXWIKAQ7coQ2chyADWherqN5vXttcJsdGp2OtwmwJU7A+Eh7\"

```

5750 ybygbm17f7t7kldwvunfr42fz1nXndtavt6mls6YcR92r6V2bZaw2pxNfMehz3EMB+mgolskzJ
d3/zNbGE1XlGPWbuXg1Lq5yCw5fBzIwgbpWnfbqgtWt4FUbwov32g9ew8LzDltMj1
aupq7t/bmxY+w/egJGk0tksyd2+g+fBbVoxD5bLzT0Rw+jb0p6U0XGOYNgR/quta3vB
fgeua6qv2d7vn8dPv3r1bdw34GSPg9i0DG9h5Xwnh9kaAmMj6dkLpZmtD3cnu7tv5C
h/Yrg1lpWpx/vruDrcD+wsq54mm+n2b2oKgyRspR41kOg21b86ytagtcPmb9v/m9LvcuATz
jow6tVpnCmxh2+snnPhSiccyJa46s7rsMyRgiw1451u01lIRw7fmlnx3z2+gfWlU2
Y572b6Eax2k9yPcJot151QnlJyLdrUfZp1/_pmkugV9n9Ag0yGUT7neivx/Ch6Ughul1uh/
f9v0ut+g703g7rf2rlx8z+w/8F6Pw6V7sXsh1nlayw2d2z1uLm+4L1mpWnA5gud9oL
ZFA66gczoxThG6Q11nR5d9ojuvaxlcvy+rFcjw3nsLkV0CefphBuICLRmlv+1K9p4ngHq6Fc2/
NcqMsIiCsnCkfExd+tMfBwuxmdPb0zG7t194225Y3TCrzPqWh0tH2raHj0/yb0kdhdpbanq2/
GKwF667/285Abh5cdpnhuJee6Y6W01g2emtMqNcDekez1tXuvC3Lk4YvTJepug5tQswFkXda
ufu9Mfwig3sqnNtcX76+3xEQWQwzveqSpvzTrm2caFy5v461+ydyd5cdxrD34U9XkeHr1oskt3g
o2ulm2ZWE0+f6R0fdtPnf2w9U207/bqz25p20l0+pdyd5cdxrD34U9XkeHr1oskt3g
AcwtK009F2N+gwtWsd0CoFdRxeNccfxWuks9CpBzXN7w+e+gw506/2014xnglrbC/
7667gRdrwH2z1MlVMyqqzttP5+7v0r1r/JL01y18+0x0eB+cv/UT05i3cNfjkS3z0M
tFuTf11+Y4faCkwzqzbyp6H1g6LjbgpwLxq90y/19jw/WX3Q3s2QgPhPr5amT1IDFnOp6/
fz5yW4FhFmxd+Bu4Nyv37y3EfB0k65ic0t+zjP+84fJq1YtGNGk7/BsT02MkAc1gjyPGL/
A4P9CxyNmDokJrEveB4HpoysBs/qyT2R6G2r160g9A8hpB4e6hsP2rztM0jeRgrBwdB2Pw
NYD1b4OSTMBrcmds2E/GG2zv/F7uejsqwy/7A7guBh6Ky19q3f3pqQvqgxX4x4d+Ueg+lmv5/
bjy+to+M5LsqppNz6nbwhfhdajemzy2ap15zld2p18yA3nqT4F3Rkyt0FaUr9Kyrl0w8
sdmc/H29o0VgtN1C+1z+tu27r+7gekbh4+83P553qooYwU/j21zWbd27xLuv0f4fa1mgQSV
2G6L+M6khvoraQwgnellzy/g1lx1+IBNCN7F207975XQfXn/u+Ha+Ur3RAg1M1Lg3B3fPyEtp
mcd5y05CzMeX9Q21AqgbgymxL9SvQsgBfUjbjBx2M+eKuerB0Rt1eBw80fj/LzYh
/Tcnsb681t7dntqN8B8Lte229w5TS1F71FsQvlytFvTu0T6b2etccB0R11e8658YeT
202udeWqmTr7w7S7d9dXv19+Lzt0Mpb73kAn4YzrdFm+5Dz5ydmYhna1ckw0PvHg5qrQs/
wCY6rwU9dkx5Mu9wQXMa+e+puqlw8/dvgf6u1LpVsPbpxsp0niQwagElsm9ggNxctoBQ1v5/
7tBBBjAdHmMdY0/q/r1f25nKQw9Aj7dsu2h12Lc182bk+1u278F7WYkf1q4/r1x2
tYvXy8bonyW6z9w+j0zw7putv1lp0NQx2Lul0vuo0D7dlyjxcrNwHeh+j0wsykrkPs
272H14lp1ciQKoyw6nMs5fysKeile0695+WCx3j5mcmj56s1+byh2YELXgkRmHny/HMKOs
aE7D34puayzeUD2WdvosJxvFse/lpz/wj0Q9e1H94WzQws62+CuVh31mtJNsfHxOrfH
/Wkqg29Fw1TCRjyJhW5t+ocSzLzQ52Bv1tG+wopQxRyeCarfRbdSg5CbD/PsyxshAkBw
pQz9y41L0u1AB44k5w6gdbH06+b0nwjzxPyAviy6e6ce00117SAZKuoxgtBzR9cavayxx
2CbmjBdAtTcruWyKiwy4myTh9zr39/8X1j0EsWetey7qFj1odwkmAfbE2Kd6l1wneH
H52HuW1alQHQOYUzrwy9tLs7ru40yQj4BwJyCayRhTye4x8/xCw+rus9L5y5c0A+
8w0v0N2zxw87ADPzCedpxpLsdxK0deFrwm+j4taEa7y2xzmj+61F2U6L46ch7e0d6/q/
WncFBtVxb6s23Nx1PvK1JbJUtbFkPbRaq1CwHuiwbiLpyKlhWbzaQyKoeMcj19y19y
Pwk79/55Bk7f5SxmcHw79y3X5y7u8ySpvBsg55hdjy/n6y6S8FygvQzL2xeOrbmwj
YkkqG552p11OK5jzsgs+2B1l84B26+/g+uoas6yuW0YljzcQucG41lq9vQY00vepu1wl1AuP4R
z2D3GL61vWe4J35kpE1kN1ub/S34rBw6XGz6f1BbbJbJ7/tLwBdrd4b6iekgpBnH
NQ73lqHbz7TEWvurXnv45R8PfPwQRnDqigVf2qB1xFEP16+rqdL82CtnVbVidBz2JFbpwMP
a93nXybqm9qXLMnmchJcnvUn5Fkm2Lb2hJ8u5m15cnx4+2rLjdQ2Njt+kYkuw01pdgcfcMz
gKgp/ahFoxv1+JtOfimzuw87FQHnAmxAdauUtxE7c7f07usUkqgyo3z3v/zoC72+sch/
Ltnplht3yWe84pGj4W4JauN7p5Xkjw841TmAbc3Q8fRlCfcJt0f0508bNa5dSFwqyfHb0
nn1d1tRGHN3eSrt+42Mk5KwTcxSfme35Jrtv0r3rm49VmQgF80iD191x6D1bvdXmkqjvb
Nfyd9x8w1mLz5MklzSe2/VzQskDpZcdyCte7lq/4B4XkFkQaNe3kL74r29f0L/g+4r/vEo
gdtTtx0u9wBkuVfmuYzLjwv1yj2pxvuu0fP0+ptFehD0+1lxxGzawxfp6e6G1z+eme2x9Lbo
0xu119p0fLaKgQhafaa5NvPhxjK7x0glUoMrn+JafFeasnLzRkXlyBf5ediw1kCk1/w7
fd+J72vEw+dPteQwKz6j2p/d5uzt+2hLhxLfnh8j7um011AjkKyvScenp1w1AlAcaZ2e
dqv2Sx/S+LnL0pde1xVtD/Skr+JL5+9vbS7z+bYNS8Q2EuPn/0a3/x1/FJzs/Vz30EcBg/
ePdTCYRCfk73g6v16p0t/XFxfwqAaVzg5t3+jbQk45+1C10+1u282f1pJmpvRvP
4yMtrk3xUwyFjGjmpt/xdbpt8sr11f1llo0f7qM3u17kFxxyqMvsFwvpg9qPzKmZdCf1h
Ylu4639w1y11N+FxC0yCvixWzlyh3n7f3i1releUtsVurmit+jkWPKyfKzZmDf2C1m
df1f6+w16/0.615m3CDD2YF12dFyqj38arBqAgsQXG1sCgcukarD0uysauvngcZ62avf/
LLGq7FpJyjy1tckPhk+n+C7+61L0jJd345+l+sdN9Y1v4rCeW9g9+Srx6t6/are2LzbX4W
tLqz7W4k+28/FffzUk1a3ky5ulm09C8NE3h9LIn15iS1yN3y8z2HsXrnRtmwvMf92t703/
j0g8vnN35zeegf1gCm1w2/fviCjoXytiel0l0xvRghMyNz1/IjtL6W3j5y8+j17dyu5/
xLJdJmu3F0xPqfGtrUgTeDf1p/xfchowAa2KvAFg5t3+jbQk45+1C10+1u282f1pJmpvRv
4yeu29wP9x1fwv/F0puypd9uNPy19n1/xNxs0n5dG59GBCw8ms31CzFkrtCQUT2CShj+wmbq
JY7X3E83m6Wq1Sl6Lb687o+ExthJ+24d2+u4z/FvsJrSt11Rk0Po00Lzntf2dZf2d12Q
8Y8V8pdsb0vHQLR0d6exvBj9y49dQpKc5m2jyj021ldv7y8f3BzQgmsDf0arTvwFtC3i
N1NQGw1JauEvqN0r728Z2vEfmdHcfUv6B6nBpR5kF1lPfPmR6H6P0s0atv0/m/8v88kM7d
CS8yEr1uSwlpLbx79264rdn2yUuN1k1ejad1uak7j0or135+1x9AqCQDFB/c7kjhkD
E5sg69K0Kh9pdRj6v3vgvEyb0dquecslvMno/oApP2k1zve8wCmJj+3k0X+3rQK8ekimw
b1xafeh93qJ8B0f8kGm659pnd5pBm1ukp2m+227RBSHk+j0PmkCtcf/Kam1s0jxtej/k
v7q/+0zmZBn/5z10H7+3nP2gNz2eyx7u120Jd6nph17/2i0z933h0LhwlyJ1reH7/2ptxev
LaycfHxSw1QikA1QikA1QikA1QikA1QikA1QikA1QikA1QikA1QikA1QikA1QikA1QikA1Q
QikA1QikA1QikA1QikA1QikA1QikA1QikA1QikA1QikA1QikA1QikA1QikA1QikA1Q
QikA1QikA1QikA1QikA1QikA1QikA1QikA1QikA1QikA1QikA1QikA1QikA1QikA1Q
QikA1QikA1QikA1QikA1QikA1QikA1QikA1QikA1QikA1QikA1QikA1QikA1QikA1Q
ba8eAv+ny7J+pt1Qbqe005C5CfcepWw5F5y/7prt10mBaZQxgh3v1RgZsd5r5H5b3h
oguWmhi71fg180+eu111xiDwLhLq1xPf+cmb9+d0h0Qon0v2M5w8hcy1n+0r2k4ujz
9gbel1fkDn21+1+Es4t1d0n5u1Re1rkvbsNaZG07QxANxM48x59m7eBo7vNzH4fet-H
mwr+h2JYsro2t1Qdpo2Pr4Y1eV1b+kxLs2oQfLm052wukbUkLpU11+2tXlwHnyFg/
wka5RlC0Jua7x6/cn9HmB5s3N6R2q1Lkwe2130s1Q0y1dja0j23s1h1n/F5RnHNTMkQsdj1
0224z9Q0uwa/wdss3PwLh7w30UD0trUqjBgeW117lywqkNc1saFaPyirB7p0usKkr81/
BtsLJQ7Hx0bWY7iW8fmu2H2k3lJq+GteB1ci+A+jT2b8tSsGg10hDy0hGcfGfUc1/x
MHO/33tgcstv/UB8PbhpwMnri0v1uw15Vp/1zd2duvS9h7p0CmNkWv4h+EzC2Fk/
Pg/17u7C7q4m1jvVbZc00x1z3H6X0Bla1s2iyay7Hsud+307KFc343iAyrr1mEtYh
afpgvZ0EtfunV4qgWn3eQ980BgtQyL9H9Hxun6t7JWfWu47vYgkLm0+2x6vKjGhuksd1
3E9K510/17Tn0a2r2t1Qdpo2Pr4Y1eV1b+kxLs2oQfLm052wukbUkLpU11+2tXlwHnyFg/
0515w/omn/rhnRWh1Qm3En5m3b67kukVc5Vpxu06FkRn6WbGfrNrpYm5zHoE10la1d
M+5B+sthfdw8v1D0AnbUy6w5f715kGv0y31e9h2ra0gB14yShsfQw9xLbhjzCbr0zL0tobx
4Bc161L6m5n4ThdWheqf1c9qBp196h96C6yBsjLkxDz7J6bdvkh0ky/WeDlWb7z2az/
tLaa065K5g20d42h2qZgEYB/1MwgMxw0yJoz5f5Yx0pYtX+3k05M1dQdnBq/vewUvLs/
ssxyFz23jCfclz1BfPqL6Lff5Fg1pBew930y/Dbst8B4xGkA8Hh1lBozYsme272pkh/
kzw1u0h21f2rsf0t8e06qHxLvd2b5j08pgrCm1zLunMgE67w75CaJw1c29+1d9aK9yMs/
0U1AGRMbmjeGxcaDofdj1LnbsB22xJppEb0PgM0D7Uf0hnRwpAlg7ooULss3t3hrkaBjhs/
Mix9hd9g/fz12T1tsUtsBkCfm4+kFns1Ldc6Pbq1xch1nQaod8va0ulq0s54v6dfudq9/
0EA8Btg/mvcrUQxm9JdSuPw72t2B4wHr4EFyj0u81c0f9w14Ep0w+EfE31200f1/
rJhnaV89nfKtV1y6h5whnW7H0qpnkGahyBp8UqP9h1yLkog1r6p2s1y10h2p17t3L7c1
n900D+wAfe1D03n12p3z2che1/g0Ed1n7Q8m5f2d1y3J1Lr6p2s1y10h2p17t3L7c1
h/B+mbyTQd24F73C1nf0w8AxwcmTlw6lq1gF2VJM2CkYXah08Lmz1w4t2k6Cg9Q1wz10x
jvnFwRkhdBfrRqwfBrykWd/pt1Qv0p9Bn/Sm171a2f1k+pt0/dlv8whp15t+dhlAxHxn/
nCAY3RkStz3K2PnBd/x1wbHg79XLz4gzs/+Y0j0oFPco2Bn/oh4aeiQ91NkDj18hDtaT2/
Bx1MqAfp+p14Q94h6Vf2h10Q+zmaVt5yHn9djs+p5w6p5f1k5zFrlxPrfVd3zKu1/
05xG5Q9waf0aGfmrC013Rw5o8Efgfj7v8r1C0x4QrXVXmkm9SmrC7pc81qkqhxWtUX/
Y06Lr5B1y2S1nYqs0x2D5w25lmo5AwckCm1sLw4t2k6Cg9Q1wz10x
Pp56Luy21CIE+q2BkjdPp2xR8whp7w11Dl0GKwv7y9m0g1LqJnLfh9Bwq7hTf6gCvEzK/
f9v0ut+g703g7rf2rlx8z+w/8F6Pw6V7sXsh1nlayw2d2z1uLm+4L1mpWnA5gud9oL
ZFA66gczoxThG6Q11nR5d9ojuvaxlcvy+rFcjw3nsLkV0CefphBuICLRmlv+1K9p4ngHq6Fc2/
NcqMsIiCsnCkfExd+tMfBwuxmdPb0zG7t194225Y3TCrzPqWh0tH2raHj0/yb0kdhdpbanq2/
GKwF667/285Abh5cdpnhuJee6Y6W01g2emtMqNcDekez1tXuvC3Lk4YvTJepug5tQswFkXda
ufu9Mfwig3sqnNtcX76+3xEQWQwzveqSpvzTrm2caFy5v461+ydyd5cdxrD34U9XkeHr1oskt3g
o2ulm2ZWE0+f6R0fdtPnf2w9U207/bqz25p20l0+pdyd5cdxrD34U9XkeHr1oskt3g
AcwtK009F2N+gwtWsd0CoFdRxeNccfxWuks9CpBzXN7w+e+gw506/2014xnglrbC/
7667gRdrwH2z1MlVMyqqzttP5+7v0r1r/JL01y18+0x0eB+cv/UT05i3cNfjkS3z0M
tFuTf11+Y4faCkwzqzbyp6H1g6LjbgpwLxq90y/19jw/WX3Q3s2QgPhPr5amT1IDFnOp6/
fz5yW4FhFmxd+Bu4Nyv37y3EfB0k65ic0t+zjP+84fJq1YtGNGk7/BsT02MkAc1gjyPGL/
A4P9CxyNmDokJrEveB4HpoysBs/qyT2R6G2r160g9A8hpB4e6hsP2rztM0jeRgrBwdB2Pw
NYD1b4OSTMBrcmds2E/GG2zv/F7uejsqwy/7A7guBh6Ky19q3f3pqQvqgxX4x4d+Ueg+lmv5/
bjy+to+M5LsqppNz6nbwhfhdajemzy2ap15zld2p18yA3nqT4F3Rkyt0FaUr9Kyrl0w8
sdmc/H29o0VgtN1C+1z+tu27r+7gekbh4+83P553qooYwU/j21zWbd27xLuv0f4fa1mgQSV
2G6L+M6khvoraQwgnellzy/g1lx1+IBNCN7F207975XQfXn/u+Ha+Ur3RAg1M1Lg3B3fPyEtp
mcd5y05CzMeX9Q21AqgbgymxL9SvQsgBfUjbjBx2M+eKuerB0Rt1eBw80fj/LzYh
/Tcnsb681t7dntqN8Lz5MklzSe2/VzQskDpZcdyCte7lq/4B4XkFkQaNe3kL74r29f0L/g+4r/vEo
wCY6rwU9dkx5Mu9wQXMa+e+puqlw8/dvgf6u1LpVsPbpxsp0niQwagElsm9ggNxctoBQ1v5/
7tBBBjAdHmMdY0/q/r1f25nKQw9Aj7dsu2h12Lc182bk+1u278F7WYkf1q4/r1x2
tYvXy8bonyW6z9w+j0zw7putv1lp0NQx2Lul0vuo0D7dlyjxcrNwHeh+j0wsykrkPs
272H14lp1ciQKoyw6nMs5fysKeile0695+WCx3j5mcmj56s1+byh2YELXgkRmHny/HMKOs
aE7D34puayzeUD2WdvosJxvFse/lpz/wj0Q9e1H94WzQws62+CuVh31mtJNsfHxOrfH
YkkqG552p11OK5jzsgs+2B1l84B26+/g+uoas6yuW0YljzcQucG41lq9vQyMvsFwv9gPzKmZdCf1w
zD3GL61vWe4J35kpE1kN1ub/S34rBw6XGz6f1BbbJbJ7/tLwBdrd4b6iekgpBnH
NQ73lqHbz7TEWvurXnv45R8PfPwQRnDqigVf2qB1xFEP16+rqdL82CtnVbVidBz2JFbpwMP
a93nXybqm9qXLMnmchJcnvUn5Fkm2Lb2hJ8u5m15cnx4+2rLjdQ2Njt+kYkuw01pdgcfcMz
gKgp/ahFoxv1+JtOfimzuw87FQHnAmxAdauUtxE7c7f07usUkqgyo3z3v/zoC72+sch/
Ltnplht3yWe84pGj4W4JauN7p5Xkjw841TmAbc3Q8fRlCfcJt0f0508bNa5dSFwqyfHb0
nn1d1tRGHN3eSrt+42Mk5KwTcxSfme35Jrtv0r3rm49VmQgF80iD191x6D1bvdXmkqjvb
Nfyd9x8w1mLz5MklzSe2/VzQskDpZcdyCte7lq/4B4XkFkQaNe3kL74r29f0L/g+4r/vEo
gdtTtx0u9wBkuVfmuYzLjwv1yj2pxvuu0fP0+ptFehD0+1lxxGzawxfp6e6G1z+eme2x9Lbo
0xu119p0fLaKgQhafaa5NvPhxjK7x0glUoMrn+JafFeasnLzRkXlyBf5ediw1kCk1/w7
fd+J72vEw+dPteQwKz6j2p/d5uzt+2hLhxLfnh8j7um011AjkKyvScenp1w1AlAcaZ2e
dqv2Sx/S+LnL0pde1xVtD/Skr+JL5+9vbS7z+bYNS8Q2EuPn/0a3/x1/FJzs/Vz30EcBg/
ePdTCYRCfk73g6v16p0t/XFxfwqAaVzg5t3+jbQk45+1C10+1u282f1pJmpvRvP
4yMtrk3xUwyFjGjmpt/xdbpt8sr11f1llo0f7qM3u17kFxxyqMvsFwv9gPzKmZdCf1w
ulu4639w1y11N+FxC0yCvixWzlyh3n7f3i1releUtsVurmit+jkWPKyfKzZmDf2d12Q
8Y8V8pdsb0vHQLR0d6exvBj9y49dQpKc5m2jyj021ldv7y8f3BzQgmsDf0arTvwFtC3i
N1NQGw1JauEvqN0r728Z2vEfmdHcfUv6B6nBpR5kF1lPfPmR6H6P0s0atv0/m/8v88kM7d
CS8yEr1uSwlpLbx79264rdn2yUuN1k1ejad1uak7j0or135+1x9AqCQDFB/c7kjhkD
E5sg69K0Kh9pdRj6v3vgvEyb0dquecslvMno/oApP2k1zve8wCmJj+3k0X+3rQK8ekimw
b1xafeh93qJ8B0f8kGm659pnd5pBm1ukp2m+227RBSHk+j0PmkCtcf/Kam1s0jxtej/k
v7q/+0zmZBn/5z10H7+3nP2gNz2eyx7u120Jd6nph17/2i0z933h0LhwlyJ1reH7/2ptxev
LaycfHxSw1QikA1QikA1QikA1QikA1QikA1QikA1QikA1QikA1QikA1QikA1QikA1QikA1Q
QikA1QikA1QikA1QikA1QikA1QikA1QikA1QikA1QikA1QikA1QikA1QikA1QikA1QikA1Q
ba8eAv+ny7J+pt1Qbqe005C5CfcepWw5F5y/7prt10mBaZQxgh3v1RgZsd5r5H5b3h
oguWmhi71fg180+eu111xiDwLhLq1xPf+cmb9+d0h0Qon0v2M5w8hcy1n+0r2k4ujz
9gbel1fkDn21+1+Es4t1d0n5u1Re1rkvbsNaZG07QxANxM48x59m7eBo7vNzH4fet-H
mwr+h2JYsro2t1Qdpo2Pr4Y1eV1b+kxLs2oQfLm052wukbUkLpU11+2tXlwHnyFg/
wka5RlC0Jua7x6/cn9HmB5s3N6R2q1Lkwe2130s1Q0y1dja0j23s1h1n/F5RnHNTMkQsdj1
0224z9Q0uwa/wdss3PwLh7w30UD0trUqjBgeW117lywqkNc1saFaPyirB7p0usKkr81/
BtsLJQ7Hx0bWY7iW8fmu2H2k3lJq+GteB1ci+A+jT2b8tSsGg10hDy0hGcfGfUc1/x
MHO/33tgcstv/UB8PbhpwMnri0v1uw15Vp/1zd2duvS9h7p0CmNkWv4h+EzC2Fk/
Pg/17u7C7q4m1jvVbZc00x1z3H6X0Bla1s2iyay7Hsud+307KFc343iAyrr1mEtYh
afpgvZ0EtfunV4qgWn3eQ980BgtQyL9H9Hxun6t7JWfWu47vYgkLm0+2x6vKjGhuksd1
3E9K510/17Tn0a2r2t1Qdpo2Pr4Y1eV1b+kxLs2oQfLm052wukbUkLpU11+2tXlwHnyFg/
0515w/omn/rhnRWh1Qm3En5m3b67kukVc5Vpxu06FkRn6WbGfrNrpYm5zHoE10la1d
M+5B+sthfdw8v1D0AnbUy6w5f715kGv0y31e9h2ra0gB14yShsfQw9xLbhjzCbr0zL0tobx
4Bc161L6m5n4ThdWheqf1c9qBp196h96C6yBsjLkxDz7J6bdvkh0ky/WeDlWb7z2az/
tLaa065K5g20d42h2qZgEYB/1MwgMxw0yJoz5f5Yx0pYtX+3k05M1dQdnBq/vewUvLs/
ssxyFz23jCfclz1BfPqL6Lff5Fg1pBew930y/Dbst8B4xGkA8Hh1lBozYsme272pkh/
kzw1u0h21f2rsf0t8e06qHxLvd2b5j08pgrCm1zLunMgE67w75CaJw1c29+1d9aK9yMs/
0U1AGRMbmjeGxcaDofdj1LnbsB22xJppEb0PgM0D7Uf0hnRwpAlg7ooULss3t3hrkaBjhs/
Mix9hd9g/fz12T1tsUtsBkCfm4+kFns1Ldc6Pbq1xch1nQaod8va0ulq0s54v6dfudq9/
0EA8Btg/mvcrUQxm9JdSuPw72t2B4wHr4EFyj0u81c0f9w14Ep0w+EfE31200f1/
rJhnaV89nfKtV1y6h5whnW7H0qpnkGahyBp8UqP9h1yLkog1r6p2s1y10h2p17t3L7c1
n900D+wAfe1D03n12p3z2che1/g0Ed1n7Q8m5f2d1y3J1Lr6p2s1y10h2p17t3L7c1
h/B+mbyTQd24F73C1nf0w8AxwcmTlw6lq1gF2VJM2CkYXah08Lmz1w4t2k6Cg9Q1wz10x
jvnFwRkhdBfrRqwfBrykWd/pt1Qv0p9Bn/Sm171a2f1k+pt0/dlv8whp15t+dhlAxHxn/
nCAY3RkStz3K2PnBd/x1wbHg79XLz4gzs/+Y0j0oFPco2Bn/oh4aeiQ91NkDj18hDtaT2/
Bx1MqAfp+p14Q94h6Vf2h10Q+zmaVt5yHn9djs+p5w6p5f1k5zFrlxPrfVd3zKu1/
05xG5Q9waf0aGfmrC013Rw5o8Efgfj7v8r1C0x4QrXVXmkm9SmrC7pc81qkqhxWtUX/
Y06Lr5B1y2S1nYqs0x2D5w25lmo5AwckCm1sLw4t2k6Cg9Q1wz10x
Pp56Luy21CIE+q2BkjdPp2xR8whp7w11Dl0GKwv7y9m0g1LqJnLfh9Bwq7hTf6gCvEzK/
f9v0ut+g703g7rf2rlx8z+w/8F6Pw6V7sXsh1nlayw2d2z1uLm+4L1mpWnA5gud9oL
ZFA66gczoxThG6Q11nR5d9ojuvaxlcvy+rFcjw3nsLkV0CefphBuICLRmlv+1K9p4ngHq6Fc2/
NcqMsIiCsnCkfExd+tMfBwuxmdPb0zG7t194225Y3TCrzPqWh0tH2raHj0/yb0kdhdpbanq2/
GKwF667/285Abh5cdpnhuJee6Y6W01g2emtMqNcDekez1tXuvC3Lk4YvTJepug5tQswFkXda
ufu9Mfwig3sqnNtcX76+3xEQWQwzveqSpvzTrm2caFy5v461+ydyd5cdxrD34U9XkeHr1oskt3g
o2ulm2ZWE0+f6R0fdtPnf2w9U207/bqz25p20l0+pdyd5cdxrD34U9XkeHr1oskt3g
AcwtK009F2N+gwtWsd0CoFdRxeNccfxWuks9CpBzXN7w+e+gw506/2014xnglrbC/
7667gRdrwH2z1MlVMyqqzttP5+7v0r1r/JL01y18+0x0eB+cv/UT05i3cNfjkS3z0M
tFuTf11+Y4faCkwzqzbyp6H1g6LjbgpwLxq90y/19jw/WX3Q3s2QgPhPr5amT1IDFnOp6/
fz5yW4FhFmxd+Bu4Nyv37y3EfB0k65ic0t+zjP+84fJq1YtGNGk7/BsT02MkAc1gjyPGL/
A4P9CxyNmDokJrEveB4HpoysBs/qyT2R6G2r160g9A8hpB4e6hsP2rztM0jeRgrBwdB2Pw
NYD1b4OSTMBrcmds2E/GG2zv/F7uejsqwy/7A7guBh6Ky19q3f3pqQvqgxX4x4d+Ueg+lmv5/
bjy+to+M5LsqppNz6nbwhfhdajemzy2ap15zld2p18yA3nqT4F3Rkyt0FaUr9Kyrl0w8
sdmc/H29o0VgtN1C+1z+tu27r+7gekbh4+83P553qooYwU/j21zWbd27xLuv0f4fa1mgQSV
2G6L+M6khvoraQwgnellzy/g1lx1+IBNCN7F207975XQfXn/u+Ha+Ur3RAg1M1Lg3B3fPyEtp
mcd5y05CzMeX9Q21AqgbgymxL9SvQsgBfUjbjBx2M+eKuerB0Rt1eBw80fj/LzYh
/Tcnsb681t7dntqN8Lz5MklzSe2/VzQskDpZcdyCte7lq/4B4XkFkQaNe3kL74r29f0L/g+4r/vEo
wCY6rwU9dkx5Mu9wQXMa+e+puqlw8/dvgf6u1LpVsPbpxsp0niQwagElsm9ggNxctoBQ1v5/
7tBBBjAdHmMdY0/q/r1f25nKQw9Aj7dsu2h12Lc182bk+1u278F7WYkf1q4/r1x2
tYvXy8bonyW6z9w+j0zw7putv1lp0NQx2Lul0vuo0D7dlyjxcrNwHeh+j0wsykrkPs
272H14lp1ciQKoyw6nMs5fysKeile0695+WCx3j5mcmj56s1+byh2YELXgkRmHny/HMKOs
aE7D34puayzeUD2WdvosJxvFse/lpz/wj0Q9e1H94WzQws62+CuVh31mtJNsfHxOrfH
YkkqG552p11OK5jzsgs+2B1l84B26+/g+uoas6yuW0YljzcQucG41lq9vQyMvsFwv9gPzKmZdCf1w
zD3GL61vWe4J35kpE1kN1ub/S34rBw6XGz6f1BbbJbJ7/tLwBdrd4b6iekgpBnH
NQ73lqHbz7TEWvurXnv45R8PfPwQRnDqigVf2qB1xFEP16+rqdL82CtnVbVidBz2JFbpwMP
a93nXybqm9qXLMnmchJcnvUn5Fkm2Lb2hJ8u5m15cnx4+2rLjdQ2Njt+kYkuw01pdgcfcMz
gKgp/ahFoxv1+JtOfimzuw87FQHnAmxAdauUtxE7c7f07usUkqgyo3z3v/zoC72+sch/
Ltnplht3yWe84pGj4W4JauN7p5Xkjw841TmAbc3Q8fRlCfcJt0f0508bNa5dSFwqyfHb0
nn1d1tRGHN3eSrt+42Mk5KwTcxSfme35Jrtv0r3rm49VmQgF80iD191x6D1bvdXmkqjvb
Nfyd9x8w1mLz5MklzSe2/VzQskDpZcdyCte7lq/4B4XkFkQaNe3kL74r29f0L/g+4r/vEo
gdtTtx0u9wBkuVfmuYzLjwv1yj2pxvuu0fP0+ptFehD0+1lxxGzawxfp6e6G1z+eme2x9Lbo
0xu119p0fLaKgQhafaa5NvPhxjK7x0glUoMrn+JafFeasnLzRkXlyBf5ediw1kCk1/w7
fd+J72vEw+dPteQwKz6j2p/d5uzt+2hLhxLfnh8j7um011AjkKyvScenp1w1AlAcaZ2e
dqv2Sx/S+LnL0pde1xVtD/Skr+JL5+9vbS7z+bYNS8Q2EuPn/0a3/x1/FJzs/Vz30EcBg/
ePdTCYRCfk73g6v16p0t/XFxfwqAaVzg5t3+jbQk45+1C10+1u282f1pJmpvRvP
4yMtrk3xUwyFjGjmpt/xdbpt8sr11f1llo0f7qM3u17kFxxyqMvsFwv9gPzKmZdCf1w
ulu4639w1y11N+FxC0yCvixWzlyh3n7f3i1releUtsVurmit+jkWPKyfKzZmDf2d12Q
8Y8V8pdsb0vHQLR0d6exvBj9y49dQpKc5m2jyj021ldv7y8f3BzQgmsDf0arTvwFtC3i
N1NQGw1JauEvqN0r728Z2vEfmdHcfUv6B6nBpR5kF1lPfPmR6H6P0s0atv0/m/8v88kM7d
CS8yEr1uSwlpLbx79264rdn2yUuN1k1ejad1uak7j0or135+1x9


```

6000 var E_FOOTER = "gsh-footer" // footer element in HTML
6001 var E_GINDEX = "gsh-gindex" // index of Golang code of GShell
6002 var E_GOCODE = "gsh-gocode" // Golang code of GSHELL
6003 var E_TODO = "gsh-todo" // TODO of GSHELL
6004 var E_DICT = "gsh-dict" // Dictionary of GSHELL
6005
6006 function bannerElem(){ return document.getElementById(E_BANNER); }
6007 function bannerStyleFunc(){ return bannerElem().style; }
6008 var bannerStyle = bannerStyleFunc()
6009 bannerStyle.backgroundImage = "url("+GShellLogo+"";
6010 //bannerStyle.backgroundImage = "url("+GShellInsideIcon+"";
6011 //bannerStyle.backgroundImage = "url("+GShellFavicon+"";
6012 GMMenu.style.backgroundImage = "url("+GShellInsideIcon+"";
6013
6014 function footerElem(){ return document.getElementById(E_FOOTER); }
6015 function footerStyle(){ return footerElem().style; }
6016 footerElem().style.backgroundImage="url("+ITSmoreQR+"";
6017 //footerStyle().backgroundImage = "url("+ITSmoreQR+"";
6018
6019 function html_fold(e){
6020   if( e.innerHTML == "Fold" ){
6021     e.innerHTML = "Unfold"
6022     document.getElementById('gsh-menu-exit').innerHTML=""
6023     document.getElementById('GshStatement').open=false
6024     GshFeatures.open = false
6025     document.getElementById('html-src').open=false
6026     document.getElementById(E_GINDEX).open=false
6027     document.getElementById(E_GOCODE).open=false
6028     document.getElementById(E_TODO).open=false
6029     document.getElementById('references').open=false
6030   }else{
6031     e.innerHTML = "Fold"
6032     document.getElementById('GshStatement').open=true
6033     GshFeatures.open = true
6034     document.getElementById(E_GINDEX).open=true
6035     document.getElementById(E_GOCODE).open=true
6036     document.getElementById(E_TODO).open=true
6037     document.getElementById('references').open=true
6038   }
6039 }
6040 function html_pure(e){
6041   if( e.innerHTML == "Pure" ){
6042     document.getElementById('gsh').style.display=true
6043     //document.style.display = false
6044     e.innerHTML = "Unpure"
6045   }else{
6046     document.getElementById('gsh').style.display=false
6047     //document.style.display = true
6048     e.innerHTML = "Pure"
6049   }
6050 }
6051
6052 var bannerIsStopping = false
6053 //NOTE: .com/JSCREF/prop_style_backgroundposition.asp
6054 function shiftBG(){
6055   bannerIsStopping = !bannerIsStopping
6056   bannerStyle.backgroundPosition = "0 0";
6057 }
6058 // status should be inherited on Window Fork(), so use the status in DOM
6059 function html_stop(e,toggle){
6060   if( toggle ){
6061     if( e.innerHTML == "Stop" ){
6062       bannerIsStopping = true
6063       e.innerHTML = "Start"
6064     }else{
6065       bannerIsStopping = false
6066       e.innerHTML = "Stop"
6067     }
6068   }else{
6069     // update JavaScript variable from DOM status
6070     if( e.innerHTML == "Stop" ){ // shown if it's running
6071       bannerIsStopping = false
6072     }else{
6073       bannerIsStopping = true
6074     }
6075   }
6076 }
6077 html_stop(document.getElementById('GshMenuStop'),false) // onInit.
6078 //html_stop(bannerElem(),false) // onInit.
6079
6080 //https://www.w3schools.com/jscrf/met_win_setinterval.asp
6081 function shiftBanner(){
6082   var now = new Date().getTime();
6083   //console.log("now="+(now%10))
6084   if( !bannerIsStopping ){
6085     bannerStyle.backgroundPosition = ((now/10)%100000)+" 0";
6086   }
6087 }
6088 setInterval(shiftBanner,10); // onInit.
6089
6090 // <a href="https://developer.mozilla.org/ja/docs/Web/API/Window/open">window.open()</a>
6091 // from embedded html to standalone page
6092 var MyChildren = 0
6093 function html_fork(){
6094   MyChildren += 1
6095   WinId = document.getElementById('gsh-WinId').innerHTML + "." + MyChildren;
6096   newwin = window.open("",WinId,"");
6097   src = document.getElementById("gsh");
6098   newwin.document.write("/*<"+html>\n");
6099   newwin.document.write("<"+span id=\"gsh\">");
6100   newwin.document.write(src.innerHTML);
6101   newwin.document.write("."+span>"+/html>\n"); // gsh span
6102   newwin.document.getElementById('gsh-menu-exit').innerHTML = "Close";
6103   newwin.document.getElementById('gsh-WinId').innerHTML = WinId;
6104   newwin.document.close();
6105   newwin.focus();
6106 }
6107 function html_close(){
6108   window.close()
6109 }
6110 function win_jump(win){
6111   //win = window.top;
6112   win = window.openner; // https://developer.mozilla.org/ja/docs/Web/API/window.opener
6113   if( win == null ){
6114     console.log("jump to window.opener(\"+win\")\n")
6115   }else{
6116     console.log("jump to window.opener(\"+win\")\n")
6117     win.focus();
6118   }
6119 }
6120
6121 // 0.2.9 2020-0902 created chekcsum of HTML
6122 CRC32UNIX = 0x04C11DB7 // Unix cksum
6123 function byteCRC32add(bigcrc,octstr,octlen){
6124   var crc = new Int32Array(1)

```

```

6125     crc[0] = bigcrc
6126
6127     let oi = 0
6128     for( ; oi < octlen; oi++ ){
6129         var oct = new Int8Array(1)
6130         oct[0] = octstr[oi]
6131         for( bi = 0; bi < 8; bi++ ){
6132             //console.log("--CRC32 "+crc[0]+"" +oct[0].toString(16)+" ["+oi+"."+bi+"]\n")
6133             ovf1 = crc[0] < 0 ? 1 : 0
6134             ovf2 = oct[0] < 0 ? 1 : 0
6135             ovf = ovf1 ^ ovf2
6136             oct[0] <= 1
6137             crc[0] <= 1
6138             if( ovf ){ crc[0] ^= CRC32UNIX }
6139         }
6140     }
6141     //console.log("--CRC32 byteAdd return crc="+crc[0]+","+oi+"/"+octlen+"\n")
6142     return crc[0];
6143 }
6144 function strCRC32add(biggcrc,stri,strlen){
6145     var crc = new Uint32Array(1)
6146     crc[0] = bigcrc
6147     var code = new Uint8Array(strlen);
6148     for( i = 0; i < strlen; i++){
6149         code[i] = stri.charCodeAt(i) // not charAt() !!!!
6150         //console.log("== "+code[i].toString(16)+" <== "+stri[i]+"\n")
6151     }
6152     crc[0] = byteCRC32add(crc,code,strlen)
6153     //console.log("--CRC32 strAdd return crc="+crc[0]+"\n")
6154     return crc[0]
6155 }
6156 function byteCRC32end(biggcrc,len){
6157     var crc = new Uint32Array(1)
6158     crc[0] = bigcrc
6159     var slen = new Uint8Array(4)
6160     let li = 0
6161     for( ; li < 4; ){
6162         selen[li] = len
6163         li += 1
6164         len >= 8
6165         if( len == 0 ){
6166             break
6167         }
6168     }
6169     crc[0] = byteCRC32add(crc[0],slen,li)
6170     crc[0] ^= 0xFFFFFFFF
6171     return crc[0]
6172 }
6173 function strCRC32(stri,len){
6174     var crc = new Uint32Array(1)
6175     crc[0] = 0
6176     crc[0] = strCRC32add(0,stri,len)
6177     crc[0] = byteCRC32end(crc[0],len)
6178     //console.log("--CRC32 "+crc[0]+" "+len+"\n")
6179     return crc[0]
6180 }
6181 function getigest(){
6182     //alert("cksum="+strCRC32("",0))
6183     //alert("cksum="+strCRC32("0",1))
6184     //return
6185
6186     version = document.getElementById('GshVersion').innerHTML
6187     sfavico = document.getElementById('GshFaviconURL').href;
6188     sbanner = document.getElementById('GshBanner').style.backgroundImage;
6189     spositi = document.getElementById('GshBanner').style.backgroundPosition;
6190     sfooter = document.getElementById('gsh-footer').style.backgroundImage;
6191     document.getElementById('GshFaviconURL').href = "";
6192     document.getElementById('GshBanner').style.backgroundImage = "";
6193     document.getElementById('GshBanner').style.backgroundPosition = "";
6194     document.getElementById('gsh-footer').style.backgroundImage = ""
6195
6196     //html = document.getElementById("gsh").outerHTML;
6197     html = document.getElementById("gsh").innerHTML;
6198
6199     textarea = document.createElement("textarea")
6200     textarea.innerHTML = html
6201     // <a href="https://stackoverflow.com/questions/5796718/html-entity-decode">Thanks</a>
6202     text = textarea.value
6203     //textarea.destroy()
6204     text = "
6205     + /*+"html>\n"           // lost preamble text
6206     + <"span id="gsh">"      // lost preamble text
6207     + text
6208     + <"/span><"html>\n" // lost trail text
6209     ;
6210
6211     tlen = text.length
6212     console.log("length="+tlen+"\n"+text)
6213     //alert("cksum : " + strCRC32(text,tlen) + " " + tlen + " " + version)
6214     digest = strCRC32(text,tlen) + " " + tlen
6215
6216     document.getElementById('GshFaviconURL').href          = sfavico;
6217     document.getElementById('GshBanner').style.backgroundImage = sbanner;
6218     document.getElementById('GshBanner').style.backgroundPosition = spositi;
6219     document.getElementById('gsh-footer').style.backgroundImage = sfooter;
6220     return digest
6221 }
6222 function html_digest(){
6223     version = document.getElementById('GshVersion').innerHTML
6224     digest = getDigest()
6225     alert("cksum: " + digest + " " + version)
6226 }
6227
6228 // source code viewer
6229 function frame_close(){
6230     srcframe = document.getElementById("src-frame");
6231     srcframe.innerHTML = "";
6232     //srcframe.style.col = 1;
6233     srcframe.style.rows = 1;
6234     srcframe.style.height = 0;
6235     srcframe.style.display = false;
6236     src = document.getElementById("src-frame-textarea");
6237     src.innerHTML = "";
6238     //src.cols = 0;
6239     src.rows = 0;
6240     src.display = false;
6241     //alert("--closed--")
6242 }
6243 //<!-- | <span onclick="html_view();">Source</span> -->
6244 //<!-- | <span onclick="frame_close();">SourceClose</span> -->
6245 //<!-- | <span>Download</span> -->
6246 function frame_open(){
6247     document.getElementById('GshFaviconURL').href = "";
6248     oldsrc = document.getElementById("GENSRC");
6249     if( oldsrc != null ){

```

```
6250     //alert("--I--(erasing old text)")
6251     oldsrc.innnerHTML = "";
6252     return
6253 }else{
6254     //alert("--I--(no old text)")
6255 }
6256 banner = document.getElementById('GshBanner').style.backgroundImage;
6257 footer = document.getElementById('gsh-footer').style.backgroundImage;
6258
6259 document.getElementById('GshFaviconURL').href = "";
6260 document.getElementById('Gstat').style = "";
6261 document.getElementById('GPos').style = "";
6262 document.getElementById('GPos').innerHTML = "";
6263 document.getElementById('GLog').style = "";
6264 document.getElementById('GLog').innerHTML = "";
6265 document.getElementById('GMenu').style = "";
6266 //document.getElementById('GMENU').style.backgroundImage = "";
6267 //document.getElementById('GMENU').style.backgroundPosition = "";
6268
6269 document.getElementById('GshBanner').style.backgroundImage = "";
6270 document.getElementById('GshBanner').style.backgroundPosition = "";
6271 document.getElementById('gsh-footer').style.backgroundImage = "";
6272 document.getElementById('rsa-oaep-message').style = "";
6273
6274 src = document.getElementById("gsh");
6275 srcframe = document.getElementById("src-frame");
6276 srcframe.innnerHTML = ""
6277 + "<"+cite id="GENSRC">\n"
6278 + "<"+style>\n"
6279 + "#GENSRC textarea{tab-size:4;}\n"
6280 + "#GENSRC textarea{-o-tab-size:4;}\n"
6281 + "#GENSRC textarea{-moz-tab-size:4;}\n"
6282 + "#GENSRC textarea{spellcheck:false;}\n"
6283 + "<"/"+style>\n"
6284 + "<"+textarea id="src-frame-textarea" cols=100 rows=20 class="gsh-code">\n"
6285 + /*<"+html>\n"           // lost preamble text
6286 + <"+span id="gsh\">>"      // lost preamble text
6287 + src.innnerHTML
6288 + "<"+span><"+/html>\n"   // lost trail text
6289 + "<"+textarea>\n"
6290 + "<"/"+cite><!-- GENSRC -->\n";
6291
6292 //srcframe.style.cols = 80;
6293 //srcframe.style.rows = 80;
6294
6295 document.getElementById('GshBanner').style.backgroundImage = banner;
6296 document.getElementById('gsh-footer').style.backgroundImage = footer;
6297 }
6298 function fill_CSSview(){
6299     part = document.getElementById('GshStyleDef')
6300     view = document.getElementById('gsh-style-view')
6301     view.innnerHTML = ""
6302     + "<"+textarea cols=100 rows=20 class="gsh-code">\n"
6303     + part.innnerHTML
6304     + "<"/"+textarea>\n"
6305 }
6306 function fill_JavaScriptView(){
6307     jspart = document.getElementById('gsh-script')
6308     view = document.getElementById('gsh-script-view')
6309     view.innnerHTML = ""
6310     + "<"+textarea cols=100 rows=20 class="gsh-code">\n"
6311     + jspart.innnerHTML
6312     + "<"/"+textarea>\n"
6313 }
6314 function fill_DataView(){
6315     part = document.getElementById('gsh-data')
6316     view = document.getElementById('gsh-data-view')
6317     view.innnerHTML = ""
6318     + "<"+textarea cols=100 rows=20 class="gsh-code">\n"
6319     + part.innnerHTML
6320     + "<"/"+textarea>\n"
6321 }
6322 function jumpTo_StyleView(){
6323     jsview = document.getElementById('html-src')
6324     jsview.open = true
6325     jsview = document.getElementById('gsh-style-frame')
6326     jsview.open = true
6327     fill_CSSview()
6328 }
6329 function jumpTo_JavaScriptView(){
6330     jsview = document.getElementById('html-src')
6331     jsview.open = true
6332     jsview = document.getElementById('gsh-script-frame')
6333     jsview.open = true
6334     fill_JavaScriptView()
6335 }
6336 function jumpTo_DataView(){
6337     jsview = document.getElementById('html-src')
6338     jsview.open = true
6339     jsview = document.getElementById('gsh-data-frame')
6340     jsview.open = true
6341     fill_DataView()
6342 }
6343 function jumpTo_WholeView(){
6344     jsview = document.getElementById('html-src')
6345     jsview.open = true
6346     jsview = document.getElementById('gsh-whole-view')
6347     jsview.open = true
6348     frame_open()
6349 }
6350 function html_view(){
6351     html_stop();
6352
6353     banner = document.getElementById('GshBanner').style.backgroundImage;
6354     footer = document.getElementById('gsh-footer').style.backgroundImage;
6355     document.getElementById('GshBanner').style.backgroundImage = "";
6356     document.getElementById('GshBanner').style.backgroundPosition = "";
6357     document.getElementById('gsh-footer').style.backgroundImage = "";
6358
6359 //srcwin = window.open("", "CodeView2","");
6360 srcwin = window.open("", "", "");
6361 srcwin.document.write("<span id="gsh\">>\n");
6362
6363 src = document.getElementById("gsh");
6364 srcwin.document.write("<"+style>\n");
6365 srcwin.document.write("textarea{tab-size:4;}\n");
6366 srcwin.document.write("textarea{-o-tab-size:4;}\n");
6367 srcwin.document.write("textarea{-moz-tab-size:4;}\n");
6368 srcwin.document.write("</style>\n");
6369 srcwin.document.write("<h2>\n");
6370 srcwin.document.write("<"+span onclick="window.close();">Close</span> | \n");
6371 //srcwin.document.write("<"+span onclick="html_stop();">Run</span>\n");
6372 srcwin.document.write("</h2>\n");
6373 srcwin.document.write("<textarea id="gsh-src-src" cols=100 rows=60>");
6374 srcwin.document.write("/<"+html>\n");
```

```
6375 srcwin.document.write("<"+span id=\"gsh\">>");
6376 srcwin.document.write(src.innerHTML);
6377 srcwin.document.write("<"+/span>"+/html>\n");
6378 srcwin.document.write("</"+textarea>\n");
6379
6380 document.getElementById('GshBanner').style.backgroundImage = banner;
6381 document.getElementById('gsh-footer').style.backgroundImage = footer
6382
6383 sty = document.getElementById("GshStyleDef");
6384 srcwin.document.write(<"+style>\n");
6385 srcwin.document.write(sty.innerHTML);
6386 srcwin.document.write("<"+/style>\n");
6387
6388 run = document.getElementById("gsh-script");
6389 srcwin.document.write(<"+script>\n");
6390 srcwin.document.write(run.innerHTML);
6391 srcwin.document.write("<"+/script>\n");
6392
6393 srcwin.document.write("<"+/span>"+/html>\n"); // gsh span
6394 srcwin.document.close();
6395 srcwin.focus();
6396 }
6397 GSH = document.getElementById("gsh")
6398
6399 //GSH.onclick = "alert('Ouch!')";
6400 //GSH.css = {"background-color:#eef;"}
6401 //GSH.style = "background-color:#eef;";
6402 //GSH.style.display = false;
6403 //alert('Ouch0!');
6404 //GSH.style.display = true;
6405
6406 // 2020-0904 created, tentative
6407 document.addEventListener('keydown',jgshCommand);
6408 //CurElement = GshStatement
6409 CurElement = GshMenu
6410 MemElement = GshMenu
6411
6412 function nextSib(e){
6413     n = e.nextSibling;
6414     for( i = 0; i < 100; i++ ){
6415         if( n == null ){
6416             break;
6417         }
6418         if( n.nodeName == "DETAILS" ){
6419             return n;
6420         }
6421         n = n.nextSibling;
6422     }
6423     return null;
6424 }
6425 function prevSib(e){
6426     n = e.previousSibling;
6427     for( i = 0; i < 100; i++ ){
6428         if( n == null ){
6429             break;
6430         }
6431         if( n.nodeName == "DETAILS" ){
6432             return n;
6433         }
6434         n = n.previousSibling;
6435     }
6436     return null;
6437 }
6438 function setColor(e,eName,eColor){
6439     if( e.hasChildNodes() ){
6440         s = e.childNodes;
6441         if( s != null ){
6442             for( ci = 0; ci < s.length; ci++ ){
6443                 if( s[ci].nodeName == eName ){
6444                     s[ci].style.color = eColor;
6445                     //s[ci].style.backgroundColor = eColor;
6446                     break;
6447                 }
6448             }
6449         }
6450     }
6451 }
6452
6453 // https://docs.microsoft.com/en-us/previous-versions//hh781509(v=vs.85)
6454 function showCurElementPosition(ev){
6455     if( document.getElementById("GPos") == null ){
6456         return;
6457     }
6458     if( GPos == null ){
6459         return;
6460     }
6461     e = CurElement
6462     y = e.getBoundingClientRect().top.toFixed(0)
6463     x = e.getBoundingClientRect().left.toFixed(0)
6464
6465     h = ev + " "
6466     h += 'y=' + y + ", " + 'x=' + x + " -- "
6467     h += "w=" + window.innerWidth + ", h=" + window.innerHeight + " -- "
6468 //GPos.test = h
6469 //GPos.innerHTML = h
6470 GPos.innerHTML = h
6471 }
6472
6473 function GShellMenu(e){
6474     d = new Date()
6475     GLog.innerHTML = "Hello, World! ("
6476     + d.getFullYear() + "/" + d.getMonth() + "/" + d.getDate() + " "
6477     + d.getHours() + ":" + d.getMinutes() + ":" + d.getSeconds()
6478     + " " + d.getMilliseconds()
6479     + " " + d.getTimezoneOffset()/60
6480     + " "
6481     + d.getTime() + "." + d.getMilliseconds() + ")"
6482
6483 }
6484 // placements of planes
6485 function GShellResizeX(ev){
6486     if( document.getElementById("GMenu") != null ){
6487         GMenu.style.left = window.innerWidth - 100
6488         GMenu.style.top = window.innerHeight - 90
6489     }
6490     GStat.style.width = window.innerWidth
6491     if( document.getElementById("GPos") != null ){
6492         GPos.style.width = window.innerWidth
6493         GPos.style.top = window.innerHeight - 30; //GPos.style.height
6494     }
6495     if( document.getElementById("GLog") != null ){
6496         GLog.style.width = window.innerWidth
6497         GLog.innerHTML = ""
6498     }
6499     if( document.getElementById("GLog") != null ){


```

```
6500     //GLog.innerHTML = "Resize: w=" + window.innerWidth +
6501     //", h=" + window.innerHeight
6502   }
6503   showCurElementPosition(ev)
6504 }
6505 function GShellResize(){
6506   GshellResizeX("[RESIZE]")
6507 }
6508 window.onresize = GShellResize
6509
6510 function ScrollToElement(oe,ne){
6511   ne.scrollIntoView()
6512   ny = ne.getBoundingClientRect().top.toFixed(0)
6513   nx = ne.getBoundingClientRect().left.toFixed(0)
6514   GLog.innerHTML = "["+ny+","+nx+"]"
6515   //window.scrollTo(0,0)
6516
6517   GTop.style.backgroundColor = "rgba(0,0,0,0.0)"
6518   GshGrid.style.left = '250px';
6519   GshGrid.style.zIndex = 0
6520   return
6521   oy = oe.getBoundingClientRect().top.toFixed(0)
6522   ox = oe.getBoundingClientRect().left.toFixed(0)
6523   y = e.getBoundingClientRect().top.toFixed(0)
6524   x = e.getBoundingClientRect().left.toFixed(0)
6525   window.scrollTo(x,y)
6526   ny = e.getBoundingClientRect().top.toFixed(0)
6527   nx = e.getBoundingClientRect().left.toFixed(0)
6528   GLog.innerHTML = "["+oy+","+ox+"]->["+y+","+x+"]->["+ny+","+nx+"]"
6529 }
6530 function jgshCommand(event){
6531   key = event
6532   keycode = key.code
6533   //GStat.style.width = window.innerWidth
6534   GStat.style.backgroundColor = "rgba(0,0,0,0.4)"
6535
6536   console.log("JSGsh-Key:"+keycode+"(^~)/")
6537   if( keycode == "Digit1" ){ // fold side-bar
6538     primary.style.width = "94%"
6539     secondary.style.width = "0%"
6540     secondary.style.opacity = 0
6541     GStat.innerHTML = "[Single Column View]"
6542   }else
6543   if( keycode == "Digit2" ){ // unfold side-bar
6544     primary.style.width = "58%"
6545     secondary.style.width = "36%"
6546     secondary.style.opacity = 1
6547     GStat.innerHTML = "[Double Column View]"
6548   }else
6549   if( keycode == "KeyU" ){ // fold/unfold all
6550     html_fold(GshMenuFold);
6551     location.href = "#"+CurElement.id;
6552   }else
6553   if( keycode == "KeyO" || keycode == "ArrowRight" ){ // fold the element
6554     CurElement.open = !CurElement.open;
6555   }else
6556   if( keycode == "ArrowRight" ){ // unfold the element
6557     CurElement.open = true
6558   }else
6559   if( keycode == "ArrowLeft" ){ // unfold the element
6560     CurElement.open = false
6561   }else
6562   if( keycode == "KeyI" ){ // inspect the element
6563     e = CurElement
6564     GLog.innerHTML = "Current Element: " + e + "<br>" +
6565       + "name='"+e.nodeName + "', "
6566       + "id='"+e.id + "', "
6567       + "children='"+e.childNodes.length + "', "
6568       + "parent='"+e.parentNode.id + "<br>" +
6569       + "text='"+e.textContent
6570     GStat.style.backgroundColor = "rgba(0,0,0,0.8)"
6571     return
6572   }else
6573   if( keycode == "KeyM" ){ // memory the position
6574     MemElement = CurElement
6575   }else
6576   if( keycode == "KeyN" || keycode == "ArrowDown" ){ // next element
6577     e = nextSib(CurElement)
6578     if( e != null ){
6579       setColor(CurElement,"SUMMARY","#fff")
6580       setColor(e,"SUMMARY","#8f8") // should be complement ?
6581       oe = CurElement
6582       CurElement = e
6583       //location.href = "#" + e.id;
6584       ScrollToElement(oe,e)
6585     }
6586   }else
6587   if( keycode == "KeyP" || keycode == "ArrowUp" ){ // previous element
6588     oe = CurElement
6589     e = prevSib(CurElement)
6590     if( e != null ){
6591       setColor(CurElement,"SUMMARY","#fff")
6592       setColor(e,"SUMMARY","#8f8") // should be complement ?
6593       CurElement = e
6594       //location.href = "#" + e.id;
6595       ScrollToElement(oe,e)
6596     }else{
6597       e = document.getElementById("GshBanner")
6598       if( e != null ){
6599         setColor(CurElement,"SUMMARY","#fff")
6600         CurElement = e
6601         ScrollToElement(oe,e)
6602       }else{
6603         e = document.getElementById("primary")
6604         if( e != null ){
6605           setColor(CurElement,"SUMMARY","#fff")
6606           CurElement = e
6607           ScrollToElement(oe,e)
6608         }
6609       }
6610     }
6611   }else
6612   if( keycode == "KeyR" ){
6613     location.reload()
6614   }else
6615   if( keycode == "KeyJ" ){
6616     GshGrid.style.top = '120px';
6617     GshGrid.innerHTML = '(>_<){Down}';
6618   }else
6619   if( keycode == "KeyK" ){
6620     GshGrid.style.top = '0px';
6621     GshGrid.innerHTML = '(-_~){Up}';
6622   }else
6623   if( keycode == "KeyH" ){
6624     GshGrid.style.left = '0px';
```

```

6625     GshGrid.innerHTML = "('_'{Left}";
6626   }else
6627     if( keycode == "KeyL" ){
6628       GLog.innerHTML +=
6629         'screen'+screen.width+'px'+'<br>' +
6630         'window'+window.innerWidth+'px'+'<br>';
6631       GshGrid.style.left = (document.documentElement.clientWidth-160).toString(10)+'px';
6632       GshGrid.innerHTML = '(@_){Right}';
6633     }else
6634     if( keycode == "KeyS" ){
6635       html_stop(GshMenuStop,true)
6636     }else
6637     if( keycode == "KeyF" ){
6638       html_fork()
6639     }else
6640     if( keycode == "KeyC" ){
6641       window.close()
6642     }else
6643     if( keycode == "KeyD" ){
6644       html_digest()
6645     }
6646
6647     showCurElementPosition("[+key.code+"] --");
6648     if( document.getElementById("GPos") != null ){
6649       //GPos.innerHTML += "[+key.code+"] --"
6650     }
6651     //GShellResizeX("[+key.code+"] --");
6652   }
6653   GShellResizeX("[INIT]");
6654 //showCurElementPosition("[INIT]");
6655   GLog.innerHTML +=
6656     '-- GShell: ' + GshVersion.innerHTML
6657     + '<br>' + '-- Digest: ' + getDigest()
6658     + '<br>' + '-- Display: ' + 'screen'+screen.width+'px', '+
6659     'window'+window.innerWidth+'px'
6660 </script>
6661
6662
6663 <!-- ##### WebCrypto ##### -->
6664 <details id="WebCrypto"><summary>WebCrypto</summary>
6665   Reference: <a href="https://mdn.github.io/dom-examples/web-crypto/encrypt-decrypt/index.html">
6666     https://mdn.github.io/dom-examples/web-crypto/encrypt-decrypt/index.html</a>
6667   <style id="web-crypto-demo-style.css">
6668     #WebCrypto *{ color:#080; font-size:9pt; }
6669     #rsa-oaep-message{ width:100% !important; height:24pt; color:#000 !important;
6670       border-width:2 !important; background-color:#f8f8f8 !important; font-size:13pt !important;
6671     }
6672     #WebCrypto input{ width:50pt; background-color:#4a4; color:#fff; border-width:0; }
6673   </style>
6674
6675   <span id="web-crypto-demo.html">
6676     <section class="encrypt-decrypt rsa-oaep">
6677       <h3 class="encrypt-decrypt-heading">Web Crypto - RSA-OAEP</h3>
6678       <section class="encrypt-decrypt-controls">
6679         <p>
6680           <b>Plain text:</b><br>
6681           <input type="textarea" id="rsa-oaep-message" name="message" value="Hello, GShell!" style="">
6682         </p>
6683         <p>
6684           <input class="encrypt-button" type="button" value="Encrypt"><br>
6685           <span class="ciphertext"><b>Cipher text:</b><br>
6686           <span class="ciphertext-value"></span></span>
6687         </p>
6688         <p>
6689           <input class="decrypt-button" type="button" value="Decrypt"><br>
6690           <span class="decrypted"><b>Decrypted text:</b><br>
6691           <span class="decrypted-value"></span></span>
6692         </p>
6693         <p>
6694           <input type="button" value="ShowKey" onclick="ShowKey()"><br>
6695           <span id="PublicKey">PublicKey...</span>
6696         </p>
6697       </section>
6698     </section>
6699   </span>
6700
6701   <script id="web-crypto-rsa-oaep.js">
6702     var RSAKeyPair = null;
6703     function ShowKey(){
6704       document.getElementById("PublicKey").innerHTML = RSAKeyPair.publicKey;
6705     }
6706     () => {
6707       //Store the calculated ciphertext here, so we can decrypt the message later.
6708       let ciphertext;
6709
6710       //Fetch the contents of the "message" textbox, and encode it
6711       //in a form we can use for the encrypt operation.
6712       function getMessageEncoding() {
6713         const messageBox = document.querySelector("#rsa-oaep-message");
6714         let message = messageBox.value;
6715         let enc = new TextEncoder();
6716         return enc.encode(message);
6717       }
6718
6719       //Get the encoded message, encrypt it and display a representation
6720       //of the ciphertext in the "Ciphertext" element.
6721       async function encryptMessage(key) {
6722         let encoded = getMessageEncoding();
6723         ciphertext = await window.crypto.subtle.encrypt(
6724           {
6725             name: "RSA-OAEP"
6726           },
6727           key,
6728           encoded
6729         );
6730
6731         //let xbuffer = new Uint8Array(ciphertext, 0, 5);
6732         let xbuffer = new Uint8Array(ciphertext, 0, ciphertext.byteLength);
6733         let b = new Uint8Array(ciphertext,0,ciphertext.byteLength);
6734         //document.write("{"+b.length+"}");
6735         //let b64 = btoa(b);
6736         let b64 = btoa(new Uint8Array(ciphertext,0,ciphertext.byteLength));
6737         const ciphertextValue = document.querySelector(".rsa-oaep .ciphertext-value");
6738         ciphertextValue.classList.add('fade-in');
6739         ciphertextValue.addEventListener('animationend', () => {
6740           ciphertextValue.classList.remove('fade-in');
6741         });
6742         ciphertextValue.textContent =
6743         ciphertext.byteLength
6744         + " bytes "
6745         + xbuffer
6746         //+ " ... "
6747         //+ b + "(" + b.length + ")"
6748         //+ b64 + "(" + b64.length + ")"
6749       ;

```

```
6750 }
6751 //Fetch the ciphertext and decrypt it.
6752 //Write the decrypted message into the "Decrypted" box.
6753 async function decryptMessage(key) {
6754     let decrypted = await window.crypto.subtle.decrypt(
6755         {
6756             name: "RSA-OAEP"
6757         },
6758         key,
6759         ciphertext
6760     );
6761
6762     let dec = new TextDecoder();
6763     const decryptedValue = document.querySelector(".rsa-oaep .decrypted-value");
6764     decryptedValue.classList.add('fade-in');
6765     decryptedValue.addEventListener('animationend', () => {
6766         decryptedValue.classList.remove('fade-in');
6767     });
6768     decryptedValue.textContent = dec.decode(decrypted);
6769 }
6770
6771 //Generate an encryption key pair, then set up event listeners
6772 //on the "Encrypt" and "Decrypt" buttons.
6773 window.crypto.subtle.generateKey(
6774     {
6775         name: "RSA-OAEP",
6776         // Consider using a 4096-bit key for systems that require long-term security
6777         modulusLength: 2048,
6778         publicExponent: new Uint8Array([1, 0, 1]),
6779         hash: "SHA-256",
6780     },
6781     true,
6782     ["encrypt", "decrypt"]
6783 ).then((keyPair) => {
6784     RSAKeyPair = keyPair
6785     const encryptButton = document.querySelector(".rsa-oaep .encrypt-button");
6786     //document.getElementById('PublicKey').innerHTML = crypto.subtle.exportKey(pkcs8, keyPair.publicKey)
6787     encryptButton.addEventListener("click", () => {
6788         encryptMessage(keyPair.publicKey);
6789     });
6790
6791     const decryptButton = document.querySelector(".rsa-oaep .decrypt-button");
6792     decryptButton.addEventListener("click", () => {
6793         decryptMessage(keyPair.privateKey);
6794     });
6795 });
6796 });
6797 })();
6798 </script>
6799 </details>
6800 */
6801 *///<br></span></html>
6802
```