

```

1  /*<html>
2  <span id="gsh">
3  <meta charset="UTF-8">
4  <meta name="viewport" content="width=device-width, initial-scale=1.0">
5  <link rel="icon" id="gsh-faviconurl" href=""/>
```

```

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>
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 = "0.3.4"
182     DATE = "2020-09-07"
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 unfind
213     Rusagev [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 GChdirHistory 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     CmdTrace bool // should be [map]
302     CmdTime 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     RSERVER 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 == '@' || ext == 'd' {
420             // IN@ . 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
472 // <a name="comexpansion">str-expansion</a>
473 // - this should be a macro processor
474 func strsubst(gshCtx *GshContext,str string,histonly bool) string {
475     rbuff := []byte{}
476     if false {
477         //@@U Unicode should be cared as a character
478         return str
479     }
480     //rstr := ""
481     inEsc := 0 // escape characer mode
482     for i := 0; i < len(str); i++ {
483         //fmt.Printf("--D--Subst %v:%v\n",i,str[i:])
484         ch := str[i]
485         if inEsc == 0 {
486             if ch == '!' {
487                 //leng,xrstr := substHistory(gshCtx,str,i,rstr)
488                 leng,rs := substHistory(gshCtx,str,i,"")
489                 if 0 < leng {
490                     //_,rs := substHistory(gshCtx,str,i,"")
491                     rbuff = append(rbuff,[]byte(rs)...)
492                     i += leng
493                     //rstr = xrstr
494                     continue
495                 }
496             }
497             switch ch {
498                 case '\\': inEsc = '\\'; continue
499                 //case '&': inEsc = '&'; continue

```

```

500         case '$':
501     }
502 }
503 switch inEsc {
504 case '\\':
505     switch ch {
506     case '\\': ch = '\\'
507     case 's': ch = ' '
508     case 't': ch = '\t'
509     case 'r': ch = '\r'
510     case 'n': ch = '\n'
511     case 'z': inEsc = 0; continue // empty, to be ignored
512     }
513     inEsc = 0
514 case '$':
515     switch {
516     case ch == '$': ch = '$'
517     case ch == 'T':
518         //rstr = rstr + time.Now().Format(time.Stamp)
519     rs := time.Now().Format(time.Stamp)
520     rbuff = append(rbuff,[]byte(rs)...)
521         inEsc = 0
522         continue;
523     default:
524         // postpone the interpretation
525         //rstr = rstr + "$" + string(ch)
526     rbuff = append(rbuff,ch)
527         inEsc = 0
528         continue;
529     }
530     inEsc = 0
531 }
532 //rstr = rstr + string(ch)
533 rbuff = append(rbuff,ch)
534 }
535 //fmt.Printf("--D--subst(%s)(%s)\n",str,string(rbuff))
536 return string(rbuff)
537 //return rstr
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 %v %s ",fi.Mode(),fi.Size(),date)
548         }
549     }
550     fmt.Printf("%s",path)
551     if isin("-sp",opts) {
552         fmt.Printf(" ")
553     }else
554     if ! isin("-n",opts) {
555         fmt.Printf("\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 }
634 // lnspl [N] [-crlf][-C \\\]
635 func (gshCtx *GshContext)SplitLine(argv[]string){
636     reader := bufio.NewReaderSize(os.Stdin,64*1024)
637     ni := 0
638     toi := 0
639     for ni = 0; ; ni++ {
640         line, err := reader.ReadString('\n')
641         if len(line) <= 0 {
642             if err != nil {
643                 fmt.Fprintf(os.Stderr,"--I-- lnspl %d to %d (%v)\n",ni,toi,err)
644                 break
645             }
646         }
647         off := 0
648         ilen := len(line)
649         remlen := len(line)
650         for oi := 0; 0 < remlen; oi++ {
651             olen := remlen
652             addnl := false
653             if 72 < olen {
654                 olen = 72
655                 addnl = true
656             }
657             fmt.Fprintf(os.Stderr,"--D-- write %d [%d.%d] %d %d/%d/%d\n",
658                 toi,ni,oi,off,olen,remlen,ilen)
659             toi += 1
660             os.Stdout.Write([]byte(line[0:olen]))
661             if addnl {
662                 //os.Stdout.Write([]byte("\r\n"))
663                 os.Stdout.Write([]byte("\n"))
664                 os.Stdout.Write([]byte("\n"))
665             }
666             line = line[olen:]
667             off += olen
668             remlen -= olen
669         }
670     }
671     fmt.Fprintf(os.Stderr,"--I-- lnspl %d to %d\n",ni,toi)
672 }
673 }
674 // CRC32 <a href="http://golang.jp/pkg/hash-crc32">crc32</a>
675 // 1 0000 0100 1100 0001 0001 1101 1011 0111
676 var CRC32UNIX uint32 = uint32(0x04C11DB7) // Unix cksum
677 var CRC32IEEE uint32 = uint32(0xEDB88320)
678 func byteCRC32add(crc uint32,str[]byte,len uint64)(uint32){
679     var oi uint64
680     for oi = 0; oi < len; oi++ {
681         var oct = str[oi]
682         for bi := 0; bi < 8; bi++ {
683             //fprintf(stderr,"--CRC32 %d %X (%d.%d)\n",crc,oct,oi,bi)
684             ovf1 := (crc & 0x80000000) != 0
685             ovf2 := (oct & 0x80) != 0
686             ovf := (ovf1 && !ovf2) || (!ovf1 && ovf2)
687             oct <<= 1
688             crc <<= 1
689             if ovf { crc ^= CRC32UNIX }
690         }
691     }
692     //fprintf(stderr,"--CRC32 return %d %d\n",crc,len)
693     return crc;
694 }
695 func byteCRC32end(crc uint32, len uint64)(uint32){
696     var slen = make([]byte,4)
697     var li = 0
698     for li = 0; li < 4; {
699         slen[li] = byte(len)
700         li += 1
701         len >>= 8
702         if( len == 0 ){
703             break
704         }
705     }
706     crc = byteCRC32add(crc,slen,uint64(li))
707     crc ^= 0xFFFFFFFF
708     return crc
709 }
710 func strCRC32(str string,len uint64)(crc uint32){
711     crc = byteCRC32add(0,[]byte(str),len)
712     crc = byteCRC32end(crc,len)
713     //fprintf(stderr,"--CRC32 %d %d\n",crc,len)
714     return crc
715 }
716 func CRC32Finish(crc uint32, table *crc32.Table, len uint64)(uint32){
717     var slen = make([]byte,4)
718     var li = 0
719     for li = 0; li < 4; {
720         slen[li] = byte(len & 0xFF)
721         li += 1
722         len >>= 8
723         if( len == 0 ){
724             break
725         }
726     }
727     crc = crc32.Update(crc,table,slen)
728     crc ^= 0xFFFFFFFF
729     return crc
730 }
731 }
732 func (gsh*GshContext)xChecksum(path string,argv[]string, sum*Checksum)(int64){
733     if isin("-type/f",argv) && !IsRegFile(path){
734         return 0
735     }
736     if isin("-type/d",argv) && IsRegFile(path){
737         return 0
738     }
739     file, err := os.OpenFile(path,os.O_RDONLY,0)
740     if err != nil {
741         fmt.Printf("--E-- cksum %v (%v)\n",path,err)
742         return -1
743     }
744     defer file.Close()
745     if gsh.CmdTrace { fmt.Printf("--I-- cksum %v %v\n",path,argv) }
746 }
747 bi := 0
748 var buff = make([]byte,32*1024)
749 var total int64 = 0

```

```

750 var initTime = time.Time{}
751 if sum.Start == initTime {
752     sum.Start = time.Now()
753 }
754 for bi = 0; ; bi++ {
755     count,err := file.Read(buff)
756     if count <= 0 || err != nil {
757         break
758     }
759     if (sum.SumType & SUM_SUM64) != 0 {
760         s := sum.Sum64
761         for _,c := range buff[0:count] {
762             s += uint64(c)
763         }
764         sum.Sum64 = s
765     }
766     if (sum.SumType & SUM_UNIXFILE) != 0 {
767         sum.Crc32Val = byteCRC32add(sum.Crc32Val,buff,uint64(count))
768     }
769     if (sum.SumType & SUM_CRCIEEE) != 0 {
770         sum.Crc32Val = crc32.Update(sum.Crc32Val,&sum.Crc32Table,buff[0:count])
771     }
772     // <a href="https://en.wikipedia.org/wiki/BSD_checksum">BSD checksum</a>
773     if (sum.SumType & SUM_SUM16_BSD) != 0 {
774         s := sum.Sum16
775         for _,c := range buff[0:count] {
776             s = (s >> 1) + ((s & 1) << 15)
777             s += int(c)
778             s &= 0xFFFF
779             //fmt.Printf("BSDsum: %d[%d] %d\n",sum.Size+int64(i),i,s)
780         }
781         sum.Sum16 = s
782     }
783     if (sum.SumType & SUM_SUM16_SYSV) != 0 {
784         for bj := 0; bj < count; bj++ {
785             sum.Sum16 += int(buff[bj])
786         }
787     }
788     total += int64(count)
789 }
790 sum.Done = time.Now()
791 sum.Files += 1
792 sum.Size += total
793 if !isin("-s",argv) {
794     fmt.Printf("%v ",total)
795 }
796 return 0
797 }
798
799 // <a name="grep">grep</a>
800 // "lines", "lin" or "lnp" for "(text) line processor" or "scanner"
801 // a*,lab,c,... sequential combination of patterns
802 // what "LINE" is should be definable
803 // generic line-by-line processing
804 // grep [-v]
805 // cat -n -v
806 // uniq [-c]
807 // tail -f
808 // sed s/x/y/ or awk
809 // grep with line count like wc
810 // rewrite contents if specified
811 func (gsh*GshContext)XGrep(path string,rexpv[]string)(int){
812     file, err := os.OpenFile(path,os.O_RDONLY,0)
813     if err != nil {
814         fmt.Printf("--E-- grep %v (%v)\n",path,err)
815         return -1
816     }
817     defer file.Close()
818     if gsh.CmdTrace { fmt.Printf("--I-- grep %v %v\n",path,rexpv) }
819     //reader := bufio.NewReaderSize(file,LINESIZE)
820     reader := bufio.NewReaderSize(file,80)
821     li := 0
822     found := 0
823     for li = 0; ; li++ {
824         line, err := reader.ReadString('\n')
825         if len(line) <= 0 {
826             break
827         }
828         if 150 < len(line) {
829             // maybe binary
830             break;
831         }
832         if err != nil {
833             break
834         }
835         if 0 <= strings.Index(string(line),rexpv[0]) {
836             found += 1
837             fmt.Printf("%s:%d: %s",path,li,line)
838         }
839     }
840     //fmt.Printf("total %d lines %s\n",li,path)
841     //if( 0 < found ){ fmt.Printf("((found %d lines %s))\n",found,path); }
842     return found
843 }
844
845 // <a name="finder">Finder</a>
846 // finding files with it name and contents
847 // file names are ORED
848 // show the content with %x fmt list
849 // ls -R
850 // tar command by adding output
851 type fileSum struct {
852     Err int64 // access error or so
853     Size int64 // content size
854     DupSize int64 // content size from hard links
855     Blocks int64 // number of blocks (of 512 bytes)
856     DupBlocks int64 // Blocks pointed from hard links
857     HLinks int64 // hard links
858     Words int64
859     Lines int64
860     Files int64
861     Dirs int64 // the num. of directories
862     SymLink int64
863     Flats int64 // the num. of flat files
864     MaxDepth int64
865     MaxNamlen int64 // max. name length
866     nextRepo time.Time
867 }
868 func showFusage(dir string,fusage *fileSum){
869     bsume := float64(((fusage.Blocks-fusage.DupBlocks)/2)*1024)/1000000.0
870     //bsumdup := float64((fusage.Blocks/2)*1024)/1000000.0
871     fmt.Printf("%v: %v files (%vd %vs %vh) %.6f MB (%.2f MBK)\n",
872         dir,
873         fusage.Files,

```

```

875     fusage.Dirs,
876     fusage.SymLink,
877     fusage.HLinks,
878     float64(fusage.Size)/1000000.0,bsume);
879 }
880 const (
881     S_IFMT      = 0170000
882     S_IFCHR     = 0020000
883     S_IFDIR     = 0040000
884     S_IFREG     = 0100000
885     S_IFLNK     = 0120000
886     S_IFSOCK    = 0140000
887 )
888 func cumFinfo(fsum *fileSum, path string, staterr error, fstat syscall.Stat_t, argv[]string,verb bool)(*fileSum){
889     now := time.Now()
890     if time.Second <= now.Sub(fsum.nextRepo) {
891         if !fsum.nextRepo.IsZero(){
892             tstamp := now.Format(time.Stamp)
893             showFusage(tstamp,fsum)
894         }
895         fsum.nextRepo = now.Add(time.Second)
896     }
897     if staterr != nil {
898         fsum.Err += 1
899         return fsum
900     }
901     fsum.Files += 1
902     if l < fstat.Nlink {
903         // must count only once...
904         // at least ignore ones in the same directory
905         //if finfo.Mode().IsRegular() {
906         if (fstat.Mode & S_IFMT) == S_IFREG {
907             fsum.HLinks += 1
908             fsum.DupBlocks += int64(fstat.Blocks)
909             //fmt.Printf("---Dup HardLink %v %s\n",fstat.Nlink,path)
910         }
911     }
912     //fsum.Size += finfo.Size()
913     fsum.Size += fstat.Size
914     fsum.Blocks += int64(fstat.Blocks)
915     //if verb { fmt.Printf("%8dBlk %s",fstat.Blocks/2,path) }
916     if isin("-ls",argv){
917         //if verb { fmt.Printf("%4d %8d ",fstat.Blksize,fstat.Blocks) }
918         // fmt.Printf("%d\t",fstat.Blocks/2)
919     }
920     //if finfo.IsDir()
921     if (fstat.Mode & S_IFMT) == S_IFDIR {
922         fsum.Dirs += 1
923     }
924     //if (finfo.Mode() & os.ModeSymLink) != 0
925     if (fstat.Mode & S_IFMT) == S_IFLNK {
926         //if verb { fmt.Printf("symlink(%v,%s)\n",fstat.Mode,finfo.Name()) }
927         //if verb { fmt.Printf("symlink(%s,%s)\n",fstat.Mode,finfo.Name()) }
928         fsum.SymLink += 1
929     }
930     return fsum
931 }
932 func (gsh*GshContext)xxFindEntv(depth int,total *fileSum,dir string, dstat syscall.Stat_t, ei int, entv []string,npatv[]string,argv[]string)(*fileSum){
933     nols := isin("-grep",argv)
934     // sort entv
935     /*
936     if isin("-t",argv){
937         sort.Slice(filev, func(i,j int) bool {
938             return 0 < filev[i].ModTime().Sub(filev[j].ModTime())
939         })
940     }
941     */
942     /*
943     if isin("-u",argv){
944         sort.Slice(filev, func(i,j int) bool {
945             return 0 < filev[i].AccTime().Sub(filev[j].AccTime())
946         })
947     }
948     if isin("-U",argv){
949         sort.Slice(filev, func(i,j int) bool {
950             return 0 < filev[i].CreateTime().Sub(filev[j].CreateTime())
951         })
952     }
953     */
954     /*
955     if isin("-S",argv){
956         sort.Slice(filev, func(i,j int) bool {
957             return filev[j].Size() < filev[i].Size()
958         })
959     }
960     */
961     for _,filename := range entv {
962         for _,npat := range npatv {
963             match := true
964             if npat == "*" {
965                 match = true
966             }else{
967                 match, _ = filepath.Match(npat,filename)
968             }
969             path := dir + DIRSEP + filename
970             if !match {
971                 continue
972             }
973             var fstat syscall.Stat_t
974             staterr := syscall.Lstat(path,&fstat)
975             if staterr != nil {
976                 if !isin("-w",argv){fmt.Printf("ufind: %v\n",staterr) }
977                 continue;
978             }
979             if isin("-du",argv) && (fstat.Mode & S_IFMT) == S_IFDIR {
980                 // should not show size of directory in "-du" mode ...
981             }else
982             if !nols && !isin("-s",argv) && (!isin("-du",argv) || isin("-a",argv)) {
983                 if isin("-du",argv) {
984                     fmt.Printf("%d\t",fstat.Blocks/2)
985                 }
986                 showFileInfo(path,argv)
987             }
988             if true { // && isin("-du",argv)
989                 total = cumFinfo(total,path,staterr,fstat,argv,false)
990             }
991             /*
992             if isin("-wc",argv) {
993             }
994             */
995             if gsh.lastCheckSum.SumType != 0 {
996                 gsh.xCksum(path,argv,&gsh.lastCheckSum);
997             }
998             x := isinX("-grep",argv); // -grep will be convenient like -ls
999             if 0 <= x && x+1 <= len(argv) { // -grep will be convenient like -ls

```



```

1000     if IsRegFile(path){
1001         found := gsh.xGrep(path,argv[x+1:])
1002         if 0 < found {
1003             foundv := gsh.CmdCurrent.FoundFile
1004             if len(foundv) < 10 {
1005                 gsh.CmdCurrent.FoundFile =
1006                     append(gsh.CmdCurrent.FoundFile,path)
1007             }
1008         }
1009     }
1010 }
1011 if !isin("-r0",argv) { // -d 0 in du, -depth n in find
1012     //total.Depth += 1
1013     if (fstat.Mode & S_IFMT) == S_IFLNK {
1014         continue
1015     }
1016     if dstat.Rdev != fstat.Rdev {
1017         fmt.Printf("--I-- don't follow differnet device %v(%v) %v(%v)\n",
1018             dir,dstat.Rdev,path,fstat.Rdev)
1019     }
1020     if (fstat.Mode & S_IFMT) == S_IFDIR {
1021         total = gsh.xxFind(depth+1,total,path,npatv,argv)
1022     }
1023 }
1024 }
1025 }
1026 return total
1027 }
1028 func (gsh*GshContext)xxFind(depth int,total *fileSum,dir string,npatv[]string,argv[]string)(*fileSum){
1029     nols := isin("-grep",argv)
1030     dirfile,oerr := os.OpenFile(dir,os.O_RDONLY,0)
1031     if oerr == nil {
1032         //fmt.Printf("--I-- %v(%v)[%d]\n",dir,dirfile,dirfile.Fd())
1033         defer dirfile.Close()
1034     }else{
1035     }
1036 }
1037 prev := *total
1038 var dstat syscall.Stat_t
1039 staterr := syscall.Lstat(dir,&dstat) // should be flstat
1040 }
1041 if staterr != nil {
1042     if !isin("-w",argv){ fmt.Printf("ufind: %v\n",staterr) }
1043     return total
1044 }
1045 //file,err := ioutil.ReadDir(dir)
1046 //_,err := ioutil.ReadDir(dir) // ReadDir() heavy and bad for huge directory
1047 /*
1048 if err != nil {
1049     if !isin("-w",argv){ fmt.Printf("ufind: %v\n",err) }
1050     return total
1051 }
1052 */
1053 if depth == 0 {
1054     total = cumPinfo(total,dir,staterr,dstat,argv,true)
1055     if !nols && !isin("-s",argv) && (!isin("-du",argv) || isin("-a",argv)) {
1056         showFileInfo(dir,argv)
1057     }
1058 }
1059 // it it is not a directory, just scan it and finish
1060 }
1061 for ei := 0; ; ei++ {
1062     entv,rderr := dirfile.Readdirnames(8*1024)
1063     if len(entv) == 0 || rderr != nil {
1064         //if rderr != nil { fmt.Printf("[%d] len=%d (%v)\n",ei,len(entv),rderr) }
1065         break
1066     }
1067     if 0 < ei {
1068         fmt.Printf("--I-- xxFind[%d] %d large-dir: %s\n",ei,len(entv),dir)
1069     }
1070     total = gsh.xxFindEntv(depth,total,dir,dstat,ei,entv,npatv,argv)
1071 }
1072 if isin("-du",argv) {
1073     // if in "du" mode
1074     fmt.Printf("%d\t%s\n",(total.Blocks-prev.Blocks)/2,dir)
1075 }
1076 return total
1077 }
1078 }
1079 // {ufind|fu|ls} [Files] [// Names] [-- Expressions]
1080 // Files is "." by default
1081 // Names is "*" by default
1082 // Expressions is "-print" by default for "ufind", or -du for "fu" command
1083 func (gsh*GshContext)xFind(argv[]string){
1084     if 0 < len(argv) && strBegins(argv[0],"?"){
1085         showFound(gsh,argv)
1086         return
1087     }
1088     if isin("-cksum",argv) || isin("-sum",argv) {
1089         gsh.lastChecksum = CheckSum{}
1090         if isin("-sum",argv) && isin("-add",argv) {
1091             gsh.lastChecksum.SumType |= SUM_SUM64
1092         }else
1093         if isin("-sum",argv) && isin("-size",argv) {
1094             gsh.lastChecksum.SumType |= SUM_SIZE
1095         }else
1096         if isin("-sum",argv) && isin("-bsd",argv) {
1097             gsh.lastChecksum.SumType |= SUM_SUM16_BSD
1098         }else
1099         if isin("-sum",argv) && isin("-sysv",argv) {
1100             gsh.lastChecksum.SumType |= SUM_SUM16_SYSV
1101         }else
1102         if isin("-sum",argv) {
1103             gsh.lastChecksum.SumType |= SUM_SUM64
1104         }
1105         if isin("-unix",argv) {
1106             gsh.lastChecksum.SumType |= SUM_UNIXFILE
1107             gsh.lastChecksum.Crc32Table = *Crc32.MakeTable(CRC32UNIX)
1108         }
1109         if isin("-ieee",argv){
1110             gsh.lastChecksum.SumType |= SUM_CRCIEEE
1111             gsh.lastChecksum.Crc32Table = *Crc32.MakeTable(CRC32IEEE)
1112         }
1113         gsh.lastChecksum.RusgAtStart = Getrusagev()
1114     }
1115     var total = fileSum{}
1116     npats := []string{}
1117     for _,v := range argv {
1118         if 0 < len(v) && v[0] != '-' {
1119             npats = append(npats,v)
1120         }
1121         if v == "/" { break }
1122         if v == "--" { break }
1123         if v == "-grep" { break }
1124         if v == "-ls" { break }

```

```

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

```

```

1250     fm := fi.Mode()
1251     if fm.IsRegular() {
1252         // R_OK=4, W_OK=2, X_OK=1, F_OK=0
1253         if syscall.Access(fullpath,5) == nil {
1254             ffullpath = fullpath
1255             ffound = true
1256             if ! isin("-s", argv) {
1257                 showFileInfo(fullpath,argv)
1258             }
1259         }
1260     }
1261 }
1262 return ffullpath, ffound
1263 }
1264 func which(list string, argv []string) (fullpathv []string, itis bool){
1265     if len(argv) <= 1 {
1266         fmt.Printf("Usage: which comand [-s] [-a] [-ls]\n")
1267         return []string{"", false}
1268     }
1269     path := argv[1]
1270     if strBegins(path, "/") {
1271         // should check if executable?
1272         ,exOK := showIfExecutable(path, "/", argv)
1273         _fmt.Printf("--D-- %v exOK=%v\n", path, exOK)
1274         return []string{path}, exOK
1275     }
1276     pathenv, efound := os.LookupEnv(list)
1277     if ! efound {
1278         fmt.Printf("--E-- which: no \"%s\" environment\n", list)
1279         return []string{"", false}
1280     }
1281     showall := isin("-a", argv) || 0 <= strings.Index(path, "*")
1282     dirv := strings.Split(pathenv, PATHSEP)
1283     ffound := false
1284     ffullpath := path
1285     for _, dir := range dirv {
1286         if 0 <= strings.Index(path, "*") { // by wild-card
1287             list, _ := ioutil.ReadDir(dir)
1288             ffullpath, ffound = showMatchFile(list, path, dir, argv)
1289         } else {
1290             ffullpath, ffound = showIfExecutable(path, dir, argv)
1291         }
1292         //if ffound && !isin("-a", argv) {
1293             if ffound && !showall {
1294                 break;
1295             }
1296         }
1297     }
1298     return []string{ffullpath}, ffound
1299 }
1300 func stripLeadingWSParg(argv []string) ([]string){
1301     for ; 0 < len(argv); {
1302         if len(argv[0]) == 0 {
1303             argv = argv[1:]
1304         } else {
1305             break
1306         }
1307     }
1308     return argv
1309 }
1310 func xEval(argv []string, nlend bool){
1311     argv = stripLeadingWSParg(argv)
1312     if len(argv) == 0 {
1313         fmt.Printf("eval [%%format] [Go-expression]\n")
1314         return
1315     }
1316     pfmt := "%v"
1317     if argv[0][0] == '$' {
1318         pfmt = argv[0]
1319         argv = argv[1:]
1320     }
1321     if len(argv) == 0 {
1322         return
1323     }
1324     gocode := strings.Join(argv, " ");
1325     //fmt.Printf("eval [%v] [%v]\n", pfmt, gocode)
1326     fset := token.NewFileSet()
1327     rval, _ := types.Eval(fset, nil, token.NoPos, gocode)
1328     fmt.Printf(pfmt, rval.Value)
1329     if nlend { fmt.Printf("\n") }
1330 }
1331 }
1332 func getval(name string) (found bool, val int) {
1333     /* should expand the name here */
1334     if name == "gsh.pid" {
1335         return true, os.Getpid()
1336     } else
1337     if name == "gsh.ppid" {
1338         return true, os.Getppid()
1339     }
1340     return false, 0
1341 }
1342 }
1343 func echo(argv []string, nlend bool){
1344     for ai := 1; ai < len(argv); ai++ {
1345         if 1 < ai {
1346             fmt.Printf(" ");
1347         }
1348         arg := argv[ai]
1349         found, val := getval(arg)
1350         if found {
1351             fmt.Printf("%d", val)
1352         } else {
1353             fmt.Printf("%s", arg)
1354         }
1355     }
1356     if nlend {
1357         fmt.Printf("\n");
1358     }
1359 }
1360 }
1361 func resfile() string {
1362     return "gsh.tmp"
1363 }
1364 //var resF *File
1365 func resmap() {
1366     //_, err := os.OpenFile(resfile(), os.O_RDWR|os.O_CREATE, os.ModeAppend)
1367     // https://deveoppaper.com/solution-to-golang-bad-file-descriptor-problem/
1368     _, err := os.OpenFile(resfile(), os.O_RDWR|os.O_CREATE, 0600)
1369     if err != nil {
1370         fmt.Printf("refF could not open: %s\n", err)
1371     } else {
1372         fmt.Printf("refF opened\n")
1373     }
1374 }

```

```

1375
1376 // @@2020-0821
1377 func gshScanArg(str string,strip int)(argv []string){
1378     var si = 0
1379     var sb = 0
1380     var inBracket = 0
1381     var argl = make([]byte,LINESIZE)
1382     var ax = 0
1383     debug := false
1384
1385     for ; si < len(str); si++ {
1386         if str[si] != ' ' {
1387             break
1388         }
1389     }
1390     sb = si
1391     for ; si < len(str); si++ {
1392         if sb <= si {
1393             if debug {
1394                 fmt.Printf("--Da- +%d %2d-%2d %s ... %s\n",
1395                     inBracket,sb,si,argl[0:ax],str[si:])
1396             }
1397         }
1398         ch := str[si]
1399         if ch == '{' {
1400             inBracket += 1
1401             if 0 < strip && inBracket <= strip {
1402                 //fmt.Printf("stripLEV %d <= %d?\n",inBracket,strip)
1403                 continue
1404             }
1405         }
1406         if 0 < inBracket {
1407             if ch == '}' {
1408                 inBracket -= 1
1409                 if 0 < strip && inBracket < strip {
1410                     //fmt.Printf("stripLEV %d < %d?\n",inBracket,strip)
1411                     continue
1412                 }
1413             }
1414             argl[ax] = ch
1415             ax += 1
1416             continue
1417         }
1418         if str[si] == ' ' {
1419             argv = append(argv,string(argl[0:ax]))
1420             if debug {
1421                 fmt.Printf("--Da- [%v][%v-%v] %s ... %s\n",
1422                     -1+len(argv),sb,si,str[sb:si],string(str[si:]))
1423             }
1424             sb = si+1
1425             ax = 0
1426             continue
1427         }
1428         argl[ax] = ch
1429         ax += 1
1430     }
1431     if sb < si {
1432         argv = append(argv,string(argl[0:ax]))
1433         if debug {
1434             fmt.Printf("--Da- [%v][%v-%v] %s ... %s\n",
1435                 -1+len(argv),sb,si,string(argl[0:ax]),string(str[si:]))
1436         }
1437     }
1438     if debug {
1439         fmt.Printf("--Da- %d [%s] => [%d]%v\n",strip,si,len(argv),argv)
1440     }
1441     return argv
1442 }
1443
1444 // should get stderr (into tmpfile ?) and return
1445 func (gsh*GshContext)Popen(name,mode string)(pin*os.File,pout*os.File,err bool){
1446     var pv = []int{-1,-1}
1447     syscall.Pipe(pv)
1448
1449     xarg := gshScanArg(name,1)
1450     name = strings.Join(xarg," ")
1451
1452     pin = os.NewFile(uintptr(pv[0]),"StdoutOf-"+name)
1453     pout = os.NewFile(uintptr(pv[1]),"StdinOf-"+name)
1454     fdix := 0
1455     dir := "?"
1456     if mode == "r" {
1457         dir = "<"
1458         fdix = 1 // read from the stdout of the process
1459     }else{
1460         dir = ">"
1461         fdix = 0 // write to the stdin of the process
1462     }
1463     gshPA := gsh.gshPA
1464     savfd := gshPA.Files[fdix]
1465
1466     var fd uintptr = 0
1467     if mode == "r" {
1468         fd = pout.Fd()
1469         gshPA.Files[fdix] = pout.Fd()
1470     }else{
1471         fd = pin.Fd()
1472         gshPA.Files[fdix] = pin.Fd()
1473     }
1474     // should do this by Goroutine?
1475     if false {
1476         fmt.Printf("--Ip- Opened fd[%v] %s %v\n",fd,dir,name)
1477         fmt.Printf("--RED1 [%d,%d,%d]->[%d,%d,%d]\n",
1478             os.Stdin.Fd(),os.Stdout.Fd(),os.Stderr.Fd(),
1479             pin.Fd(),pout.Fd(),pout.Fd())
1480     }
1481     savi := os.Stdin
1482     savo := os.Stdout
1483     save := os.Stderr
1484     os.Stdin = pin
1485     os.Stdout = pout
1486     os.Stderr = pout
1487     gsh.BackGround = true
1488     gsh.gshellh(name)
1489     gsh.BackGround = false
1490     os.Stdin = savi
1491     os.Stdout = savo
1492     os.Stderr = save
1493
1494     gshPA.Files[fdix] = savfd
1495     return pin,pout,false
1496 }
1497
1498 // <a name="ex-commands">External commands</a>
1499 func (gsh*GshContext)excommand(exec bool, argv []string) (notf bool,exit bool) {

```

```

1500     if gsh.CmdTrace { fmt.Printf("--I-- excommand[%v](%v)\n",exec,argv) }
1501
1502     gshPA := gsh.gshPA
1503     fullpathv, itis := which("PATH",[]string{"which",argv[0],"-s"})
1504     if itis == false {
1505         return true,false
1506     }
1507     fullpath := fullpathv[0]
1508     argv = unescapeWhiteSPV(argv)
1509     if 0 < strings.Index(fullpath,".go") {
1510         nargv := argv // []string{}
1511         gofullpathv, itis := which("PATH",[]string{"which","go","-s"})
1512         if itis == false {
1513             fmt.Printf("--F-- Go not found\n")
1514             return false,true
1515         }
1516         gofullpath := gofullpathv[0]
1517         nargv = []string{ gofullpath, "run", fullpath }
1518         fmt.Printf("--I-- %s (%s %s %s)\n",gofullpath,
1519             nargv[0],nargv[1],nargv[2])
1520         if exec {
1521             syscall.Exec(gofullpath,nargv,os.Environ())
1522         }else{
1523             pid, _ := syscall.ForkExec(gofullpath,nargv,&gshPA)
1524             if gsh.BackGround {
1525                 fmt.Fprintf(stderr,"--Ip- in Background pid[%d]&d(%v)\n",pid,len(argv),nargv)
1526                 gsh.BackGroundJobs = append(gsh.BackGroundJobs,pid)
1527             }else{
1528                 rusage := syscall.Rusage {}
1529                 syscall.Wait4(pid,nil,0,&rusage)
1530                 gsh.LastRusage = rusage
1531                 gsh.CmdCurrent.Rusagev[1] = rusage
1532             }
1533         }
1534     }else{
1535         if exec {
1536             syscall.Exec(fullpath,argv,os.Environ())
1537         }else{
1538             pid, _ := syscall.ForkExec(fullpath,argv,&gshPA)
1539             //fmt.Printf("[%d]\n",pid); // '&' to be background
1540             if gsh.BackGround {
1541                 fmt.Fprintf(stderr,"--Ip- in Background pid[%d]&d(%v)\n",pid,len(argv),argv)
1542                 gsh.BackGroundJobs = append(gsh.BackGroundJobs,pid)
1543             }else{
1544                 rusage := syscall.Rusage {}
1545                 syscall.Wait4(pid,nil,0,&rusage);
1546                 gsh.LastRusage = rusage
1547                 gsh.CmdCurrent.Rusagev[1] = rusage
1548             }
1549         }
1550     }
1551     return false,false
1552 }
1553
1554 // <a name="builtin">Builtin Commands</a>
1555 func (gshCtx *GshContext) sleep(argv []string) {
1556     if len(argv) < 2 {
1557         fmt.Printf("Sleep 100ms, 100us, 100ns, ...)\n")
1558         return
1559     }
1560     duration := argv[1];
1561     d, err := time.ParseDuration(duration)
1562     if err != nil {
1563         d, err = time.ParseDuration(duration+"s")
1564         if err != nil {
1565             fmt.Printf("duration ? %s (%s)\n",duration,err)
1566             return
1567         }
1568     }
1569     //fmt.Printf("Sleep %v\n",duration)
1570     time.Sleep(d)
1571     if 0 < len(argv[2:]) {
1572         gshCtx.gshelly(argv[2:])
1573     }
1574 }
1575 func (gshCtx *GshContext)repeat(argv []string) {
1576     if len(argv) < 2 {
1577         return
1578     }
1579     start0 := time.Now()
1580     for ri, _ := strconv.Atoi(argv[1]); 0 < ri; ri-- {
1581         if 0 < len(argv[2:]) {
1582             //start := time.Now()
1583             gshCtx.gshelly(argv[2:])
1584             end := time.Now()
1585             elps := end.Sub(start0);
1586             if( 1000000000 < elps ){
1587                 fmt.Printf("(repeat#%d %v)\n",ri,elps);
1588             }
1589         }
1590     }
1591 }
1592
1593 func (gshCtx *GshContext)gen(argv []string) {
1594     gshPA := gshCtx.gshPA
1595     if len(argv) < 2 {
1596         fmt.Printf("Usage: %s N\n",argv[0])
1597         return
1598     }
1599     // should br repeated by "repeat" command
1600     count, _ := strconv.Atoi(argv[1])
1601     fd := gshPA.Files[1] // Stdout
1602     file := os.NewFile(fd,"internalStdOut")
1603     fmt.Printf("--I-- Gen. Count=%d to [%d]\n",count,file.Fd())
1604     //buf := []byte{}
1605     outdata := "0123 5678 0123 5678 0123 5678 0123 5678\r"
1606     for gi := 0; gi < count; gi++ {
1607         file.WriteString(outdata)
1608     }
1609     //file.WriteString("\n")
1610     fmt.Printf("\n(%d B)\n",count*len(outdata));
1611     //file.Close()
1612 }
1613
1614 // <a name="rexec">Remote Execution</a> // 2020-0820
1615 func Elapsed(from time.Time)(string){
1616     elps := time.Now().Sub(from)
1617     if 1000000000 < elps {
1618         return fmt.Sprintf("[%5d.%02ds]",elps/1000000000,(elps%1000000000)/1000000)
1619     }else
1620     if 1000000 < elps {
1621         return fmt.Sprintf("[%3d.%03dms]",elps/1000000,(elps%1000000)/1000)
1622     }else{
1623         return fmt.Sprintf("[%3d.%03dus]",elps/1000,(elps%1000))
1624     }
}

```

```

1625 }
1626 func abftime(nanos int64)(string){
1627     if 1000000000 < nanos {
1628         return fmt.Sprintf("%d.%02ds",nanos/1000000000,(nanos%1000000000)/10000000)
1629     }else
1630     if 1000000 < nanos {
1631         return fmt.Sprintf("%d.%03dms",nanos/1000000,(nanos%1000000)/1000)
1632     }else{
1633         return fmt.Sprintf("%d.%03dus",nanos/1000,(nanos%1000))
1634     }
1635 }
1636 func absfsize(size int64)(string){
1637     fsize := float64(size)
1638     if 1024*1024*1024 < size {
1639         return fmt.Sprintf("%.2fGiB",fsize/(1024*1024*1024))
1640     }else
1641     if 1024*1024 < size {
1642         return fmt.Sprintf("%.3fMiB",fsize/(1024*1024))
1643     }else{
1644         return fmt.Sprintf("%.3fKiB",fsize/1024)
1645     }
1646 }
1647 func absfsize(size int64)(string){
1648     fsize := float64(size)
1649     if 1024*1024*1024 < size {
1650         return fmt.Sprintf("%.2fGiB",fsize/(1024*1024*1024))
1651     }else
1652     if 1024*1024 < size {
1653         return fmt.Sprintf("%.3fMiB",fsize/(1024*1024))
1654     }else{
1655         return fmt.Sprintf("%.3fKiB",fsize/1024)
1656     }
1657 }
1658 func absfspeed(totalB int64,ns int64)(string){
1659     MBs := (float64(totalB)/1000000) / (float64(ns)/1000000000)
1660     if 1000 <= MBs {
1661         return fmt.Sprintf("%.3fGB/s",MBs/1000)
1662     }
1663     if 1 <= MBs {
1664         return fmt.Sprintf("%.3fMB/s",MBs)
1665     }else{
1666         return fmt.Sprintf("%.3fKB/s",MBs*1000)
1667     }
1668 }
1669 func absfspeed(totalB int64,ns time.Duration)(string){
1670     MBs := (float64(totalB)/1000000) / (float64(ns)/1000000000)
1671     if 1000 <= MBs {
1672         return fmt.Sprintf("%.3fGBps",MBs/1000)
1673     }
1674     if 1 <= MBs {
1675         return fmt.Sprintf("%.3fMBps",MBs)
1676     }else{
1677         return fmt.Sprintf("%.3fKBps",MBs*1000)
1678     }
1679 }
1680 func fileRelay(what string,in*os.File,out*os.File,size int64,bsiz int)(wcount int64){
1681     Start := time.Now()
1682     buff := make([]byte,bsiz)
1683     var total int64 = 0
1684     var rem int64 = size
1685     nio := 0
1686     Prev := time.Now()
1687     var PrevSize int64 = 0
1688
1689     fmt.Printf(Elapsed(Start)+"--In- X: %s (%v/%v/%v) START\n",
1690         what,absfsize(total),size,nio)
1691
1692     for i:= 0; ; i++ {
1693         var len = bsiz
1694         if int(rem) < len {
1695             len = int(rem)
1696         }
1697         Now := time.Now()
1698         Elps := Now.Sub(Prev);
1699         if 1000000000 < Now.Sub(Prev) {
1700             fmt.Printf(Elapsed(Start)+"--In- X: %s (%v/%v/%v) %s\n",
1701                 what,absfsize(total),size,nio,
1702                 absfspeed((total-PrevSize),Elps))
1703             Prev = Now;
1704             PrevSize = total
1705         }
1706         rlen := len
1707         if in != nil {
1708             // should watch the disconnection of out
1709             rcc,err := in.Read(buff[0:rlen])
1710             if err != nil {
1711                 fmt.Printf(Elapsed(Start)+"--En- X: %s read(%v,%v)<%v\n",
1712                     what,rcc,err,in.Name())
1713                 break
1714             }
1715             rlen = rcc
1716             if string(buff[0:10]) == "(SoftEOF " {
1717                 var ecc int64 = 0
1718                 fmt.Sscanf(string(buff),"(SoftEOF %v",&ecc)
1719                 fmt.Printf(Elapsed(Start)+"--En- X: %s Recv ((SoftEOF %v))%v\n",
1720                     what,ecc,total)
1721                 if ecc == total {
1722                     break
1723                 }
1724             }
1725         }
1726
1727         wlen := rlen
1728         if out != nil {
1729             wcc,err := out.Write(buff[0:wlen])
1730             if err != nil {
1731                 fmt.Printf(Elapsed(Start)+"--En-- X: %s write(%v,%v)>%v\n",
1732                     what,wcc,err,out.Name())
1733                 break
1734             }
1735             wlen = wcc
1736         }
1737         if wlen < rlen {
1738             fmt.Printf(Elapsed(Start)+"--En- X: %s incomplete write (%v/%v)\n",
1739                 what,wlen,rlen)
1740             break;
1741         }
1742
1743         nio += 1
1744         total += int64(rlen)
1745         rem -= int64(rlen)
1746         if rem <= 0 {
1747             break
1748         }
1749     }

```

```

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

```

```

1875 // upload {srcfile|-zN} [dstfile]
1876 var dsize int64 = 32*1024*1024
1877 var bsize int = 64*1024
1878 var fname string = ""
1879 var out *os.File = nil
1880 if 1 < len(cmdv) { // localfile
1881     fmt.Sscanf(cmdv[1],"%d",&dsize)
1882 }
1883 if 2 < len(cmdv) {
1884     fname = cmdv[2]
1885     if fname == "" {
1886         // nul dev
1887     }else{
1888         if strBegins(fname,"{") {
1889             xin,xout,err := gsh.Popen(fname,"w")
1890             if err {
1891                 }else{
1892                     xin.Close()
1893                     defer xout.Close()
1894                     out = xout
1895                 }
1896             }else{
1897                 // should write to temporary file
1898                 // should suppress ^C on tty
1899 xout,err := os.OpenFile(fname,os.O_CREATE|os.O_RDWR|os.O_TRUNC,0600)
1900 //fmt.Printf("--In- S: open(%v) out(%v) err(%v)\n",fname,xout,err)
1901         if err != nil {
1902             fmt.Printf("--En- PUT (%v)\n",err)
1903         }else{
1904             out = xout
1905         }
1906     }
1907     fmt.Printf(Elapsed(Start)+"--In- L: open(%v,w) %v (%v)\n",
1908         fname,local,err)
1909 }
1910     fmt.Printf(Elapsed(Start)+"--In- PUT %v (/%v)\n",dsize,bsize)
1911     fmt.Printf(Elapsed(Start)+"--In- S: 200 %v OK\r\n",dsize)
1912     fmt.Fprintf(clnt,"200 %v OK\r\n",dsize)
1913     fileRelay("RecvPUT",clnt,out,dsize,bsize)
1914     res = fmt.Sprintf("200 PUT done\r\n")
1915 default:
1916     res = fmt.Sprintf("400 What? %v",req)
1917 }
1918 swcc,serr := clnt.Write([]byte(res))
1919 if serr != nil {
1920     fmt.Printf(Elapsed(Start)+"--In- S: (wc=%v er=%v) %v",swcc,serr,res)
1921 }else{
1922     fmt.Printf(Elapsed(Start)+"--In- S: %v",res)
1923 }
1924 aconn.Close();
1925 clnt.Close();
1926 }
1927 sconn.Close();
1928 }
1929 func (gsh*GshContext)RexecClient(argv []string)(int,string){
1930     debug := true
1931     Start := time.Now()
1932     if len(argv) == 1 {
1933         return -1,"EmptyARG"
1934     }
1935     argv = argv[1:]
1936     if argv[0] == "-serv" {
1937         gsh.RexecServer(argv[1:])
1938         return 0,"Server"
1939     }
1940     remote := "0.0.0.0:9999"
1941     if argv[0][0] == '8' {
1942         remote = argv[0][1:]
1943         argv = argv[1:]
1944     }
1945     if argv[0] == "-s" {
1946         debug = false
1947         argv = argv[1:]
1948     }
1949     dport, err := net.ResolveTCPAddr("tcp",remote);
1950     if err != nil {
1951         fmt.Printf(Elapsed(Start)+"Address error: %s (%s)\n",remote,err)
1952         return -1,"AddressError"
1953     }
1954     fmt.Printf(Elapsed(Start)+"--In- C: Connecting to %s\n",remote)
1955     serv, err := net.DialTCP("tcp",nil,dport)
1956     if err != nil {
1957         fmt.Printf(Elapsed(Start)+"Connection error: %s (%s)\n",remote,err)
1958         return -1,"CannotConnect"
1959     }
1960     if debug {
1961         al := serv.LocalAddr()
1962         fmt.Printf(Elapsed(Start)+"--In- C: Connected to %v <- %v\n",remote,al)
1963     }
1964     req := ""
1965     res := make([]byte,LINESIZE)
1966     count,err := serv.Read(res)
1967     if err != nil {
1968         fmt.Printf("--En- S: (%3d,%v) %v",count,err,string(res))
1969     }
1970     if debug { fmt.Printf(Elapsed(Start)+"--In- S: %v",string(res)) }
1971 }
1972 if argv[0] == "GET" {
1973     savPA := gsh.gshPA
1974     var bsize int = 64*1024
1975     req = fmt.Sprintf("%v\r\n",strings.Join(argv," "))
1976     fmt.Printf(Elapsed(Start)+"--In- C: %v",req)
1977     fmt.Fprintf(serv,req)
1978     count,err = serv.Read(res)
1979     if err != nil {
1980     }else{
1981         var dsize int64 = 0
1982         var out *os.File = nil
1983         var out_tobeclosed *os.File = nil
1984         var fname string = ""
1985         var rcode int = 0
1986         var pid int = -1
1987         fmt.Sscanf(string(res),"%d %d",&rcode,&dsize)
1988         fmt.Printf(Elapsed(Start)+"--In- S: %v",string(res[0:count]))
1989         if 3 <= len(argv) {
1990             fname = argv[2]
1991             if strBegins(fname,"{") {
1992                 xin,xout,err := gsh.Popen(fname,"w")
1993                 if err {
1994                     }else{
1995                         xin.Close()
1996                         defer xout.Close()
1997                         out = xout
1998                         out_tobeclosed = xout
1999                     }

```



```

2000         pid = 0 // should be its pid
2001     }
2002 }else{
2003     // should write to temporary file
2004     // should suppress ^C on tty
2005     xout,err := os.OpenFile(fname,os.O_CREATE|os.O_RDWR|os.O_TRUNC,0600)
2006     if err != nil {
2007         fmt.Printf("--En- %v\n",err)
2008     }
2009     out = xout
2010     //fmt.Printf("--In-- %d > %s\n",out.Fd(),fname)
2011 }
2012 }
2013 in, _ := serv.File()
2014 fileRelay("RecvGET",in,out,dsize,bsize)
2015 if 0 <= pid {
2016     gsh.gshPA = savPA // recovery of Fd(), and more?
2017     fmt.Printf(Elapsed(Start)+"--In- L: close Pipe > %v\n",fname)
2018     out_tobeClosed.Close()
2019     //syscall.Wait4(pid,nil,0,nil) //@@
2020 }
2021 }
2022 }else
2023 if argv[0] == "PUT" {
2024     remote, _ := serv.File()
2025     var local *os.File = nil
2026     var dsize int64 = 32*1024*1024
2027     var bsize int = 64*1024
2028     var ofile string = "-"
2029     //fmt.Printf("--I-- Rex %v\n",argv)
2030     if 1 < len(argv) {
2031         fname := argv[1]
2032         if strBegins(fname,"-z") {
2033             fmt.Sscanf(fname[2:], "%d",&dsize)
2034         }else
2035         if strBegins(fname,"") {
2036             xin,xout,err := gsh.Popen(fname,"r")
2037             if err {
2038                 }else{
2039                 xout.Close()
2040                 defer xin.Close()
2041                 //in = xin
2042                 local = xin
2043                 fmt.Printf("--In- [%d] < Upload output of %v\n",
2044                     local.Fd(),fname)
2045                 ofile = "-from."+fname
2046                 dsize = MaxStreamSize
2047             }
2048         }else{
2049             xlocal,err := os.Open(fname)
2050             if err != nil {
2051                 fmt.Printf("--En- (%s)\n",err)
2052                 local = nil
2053             }else{
2054                 local = xlocal
2055                 fi, _ := local.Stat()
2056                 dsize = fi.Size()
2057                 defer local.Close()
2058                 //fmt.Printf("--I-- Rex in(%v / %v)\n",ofile,dsize)
2059             }
2060             ofile = fname
2061             fmt.Printf(Elapsed(Start)+"--In- L: open(%v,r)=%v %v (%v)\n",
2062                 fname,dsize,local,err)
2063         }
2064     }
2065     if 2 < len(argv) && argv[2] != "" {
2066         ofile = argv[2]
2067         //fmt.Printf("(%d)%v B.ofile=%v\n",len(argv),argv,ofile)
2068     }
2069     //fmt.Printf(Elapsed(Start)+"--I-- Rex out(%v)\n",ofile)
2070     fmt.Printf(Elapsed(Start)+"--In- PUT %v (/%v)\n",dsize,bsize)
2071     req = fmt.Sprintf("PUT %v %v \r\n",dsize,ofile)
2072     if debug { fmt.Printf(Elapsed(Start)+"--In- C: %v",req) }
2073     fmt.Fprintf(serv,"%v",req)
2074     count,err = serv.Read(res)
2075     if debug { fmt.Printf(Elapsed(Start)+"--In- S: %v",string(res[0:count])) }
2076     fileRelay("SendPUT",local,remote,dsize,bsize)
2077 }else{
2078     req = fmt.Sprintf("%v\r\n",strings.Join(argv, " "))
2079     if debug { fmt.Printf(Elapsed(Start)+"--In- C: %v",req) }
2080     fmt.Fprintf(serv,"%v",req)
2081     //fmt.Printf("--In- sending RexRequest(%v)\n",len(req))
2082 }
2083 //fmt.Printf(Elapsed(Start)+"--In- waiting RexResponse...\n")
2084 count,err = serv.Read(res)
2085 ress := ""
2086 if count == 0 {
2087     ress = "(nil)\r\n"
2088 }else{
2089     ress = string(res[:count])
2090 }
2091 if err != nil {
2092     fmt.Printf(Elapsed(Start)+"--En- S: (%d,%v) %v",count,err,ress)
2093 }else{
2094     fmt.Printf(Elapsed(Start)+"--In- S: %v",ress)
2095 }
2096 serv.Close()
2097 //conn.Close()
2098
2099 var stat string
2100 var rcode int
2101 fmt.Sscanf(ress,"%d %s",&rcode,&stat)
2102 //fmt.Printf("--D-- Client: %v (%v)",rcode,stat)
2103 return rcode,ress
2104 }
2105 }
2106 // <a name="remote-sh">Remote Shell</a>
2107 // gcp file [...] { [host]:[port]:[dir] | dir } // -p | -no-p
2108 func (gsh*GshContext)FileCopy(argv []string){
2109     var host = ""
2110     var port = ""
2111     var upload = false
2112     var download = false
2113     var xargv = []string{"rex-gcp"}
2114     var srcv = []string{}
2115     var dstv = []string{}
2116     argv = argv[1:]
2117 }
2118 for _,v := range argv {
2119     /*
2120     if v[0] == '-' { // might be a pseudo file (generated date)
2121         continue
2122     }
2123     */
2124     obj := strings.Split(v,":")

```

```

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

```

```

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

```

```

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

```

```

2500     }
2501     if 0 <= atWorkDirX {
2502         if v.WorkDirX != atWorkDirX {
2503             continue
2504         }
2505     }
2506     if !isin("-n",argv){ // like "fc"
2507         fmt.Printf("!\%-2d ",i)
2508     }
2509     if isin("-v",argv){
2510         fmt.Println(v) // should be with it date
2511     }else{
2512         if isin("-l",argv) || isin("-l0",argv) {
2513             elps := v.EndAt.Sub(v.StartAt);
2514             start := v.StartAt.Format(time.Stamp)
2515             fmt.Printf("@%d ",v.WorkDirX)
2516             fmt.Printf("[%v] %11v/t ",start,elps)
2517         }
2518         if isin("-l",argv) && !isin("-l0",argv){
2519             fmt.Printf("%v",Rusagef("%t %u\t// %s",argv,v.Rusagev))
2520         }
2521         if isin("-at",argv) { // !isin("-ls",argv){
2522             dhi := v.WorkDirX // workdir history index
2523             fmt.Printf("@%d %s\t",dhi,v.WorkDir)
2524             // show the FileInfo of the output command??
2525         }
2526         fmt.Printf("%s",v.CmdLine)
2527         fmt.Printf("\n")
2528     }
2529 }
2530 }
2531 // ln - history index
2532 func searchHistory(gshCtx GshContext, gline string) (string, bool, bool){
2533     if gline[0] == '!' {
2534         hix, err := strconv.Atoi(gline[1:])
2535         if err != nil {
2536             fmt.Printf("--E-- (%s : range)\n",hix)
2537             return "", false, true
2538         }
2539         if hix < 0 || len(gshCtx.CommandHistory) <= hix {
2540             fmt.Printf("--E-- (%d : out of range)\n",hix)
2541             return "", false, true
2542         }
2543         return gshCtx.CommandHistory[hix].CmdLine, false, false
2544     }
2545     // search
2546     //for i, v := range gshCtx.CommandHistory {
2547     //}
2548     return gline, false, false
2549 }
2550 func (gsh*GshContext)cmdStringInHistory(hix int)(cmd string, ok bool){
2551     if 0 <= hix && hix < len(gsh.CommandHistory) {
2552         return gsh.CommandHistory[hix].CmdLine,true
2553     }
2554     return "",false
2555 }
2556 }
2557 // temporary adding to PATH environment
2558 // cd name -lib for LD_LIBRARY_PATH
2559 // chdir with directory history (date + full-path)
2560 // -s for sort option (by visit date or so)
2561 func (gsh*GshContext)ShowChdirHistory1(i int,v CChdirHistory, argv []string){
2562     fmt.Printf("!\%-2d ",v.CmdIndex) // the first command at this WorkDir
2563     fmt.Printf("@%d ",i)
2564     fmt.Printf("[%v] ",v.MovedAt.Format(time.Stamp))
2565     showFileInfo(v.Dir,argv)
2566 }
2567 func (gsh*GshContext)ShowChdirHistory(argv []string){
2568     for i, v := range gsh.CchdirHistory {
2569         gsh.ShowChdirHistory1(i,v,argv)
2570     }
2571 }
2572 func skipOpts(argv[]string)(int){
2573     for i,v := range argv {
2574         if strBegins(v,"-") {
2575             }else{
2576                 return i
2577             }
2578     }
2579     return -1
2580 }
2581 func (gshCtx*GshContext)xChdir(argv []string){
2582     cdhist := gshCtx.CchdirHistory
2583     if isin("? ",argv) || isin("-t",argv) || isin("-a",argv) {
2584         gshCtx.ShowChdirHistory(argv)
2585         return
2586     }
2587     pwd, _ := os.Getwd()
2588     dir := ""
2589     if len(argv) <= 1 {
2590         dir = toFullPath("-")
2591     }else{
2592         i := skipOpts(argv[1:])
2593         if i < 0 {
2594             dir = toFullPath("-")
2595         }else{
2596             dir = argv[1+i]
2597         }
2598     }
2599     if strBegins(dir,"@") {
2600         if dir == "@0" { // obsolete
2601             dir = gshCtx.StartDir
2602         }else
2603         if dir == "@1" {
2604             index := len(cdhist) - 1
2605             if 0 < index { index -= 1 }
2606             dir = cdhist[index].Dir
2607         }else{
2608             index, err := strconv.Atoi(dir[1:])
2609             if err != nil {
2610                 fmt.Printf("--E-- xChdir(%v)\n",err)
2611                 dir = "?"
2612             }else
2613             if len(gshCtx.CchdirHistory) <= index {
2614                 fmt.Printf("--E-- xChdir(history range error)\n")
2615                 dir = "?"
2616             }else{
2617                 dir = cdhist[index].Dir
2618             }
2619         }
2620     }
2621     if dir != "?" {
2622         err := os.Chdir(dir)
2623         if err != nil {
2624             fmt.Printf("--E-- xChdir(%s)(%v)\n",argv[1],err)

```

```

2625     }else{
2626         cwd, _ := os.Getwd()
2627         if cwd != pwd {
2628             hist1 := GChdirHistory { }
2629             hist1.Dir = cwd
2630             hist1.MovedAt = time.Now()
2631             hist1.CmdIndex = len(gshCtx.CommandHistory)+1
2632             gshCtx.ChdirHistory = append(cdhist,hist1)
2633             if !isin("-s",argv){
2634                 //cwd, _ := os.Getwd()
2635                 //fmt.Printf("%s\n",cwd)
2636                 ix := len(gshCtx.ChdirHistory)-1
2637                 gshCtx.ShowChdirHistory1(ix,hist1,argv)
2638             }
2639         }
2640     }
2641 }
2642 if isin("-ls",argv){
2643     cwd, _ := os.Getwd()
2644     showFileInfo(cwd,argv);
2645 }
2646 }
2647 func TimeValSub(tv1 *syscall.Timeval, tv2 *syscall.Timeval){
2648     *tv1 = syscall.NsecToTimeval(tv1.Nano() - tv2.Nano())
2649 }
2650 func RusageSubv(rul, ru2 [2]syscall.Rusage) ([2]syscall.Rusage){
2651     TimeValSub(&rul[0].Utime,&ru2[0].Utime)
2652     TimeValSub(&rul[0].Stime,&ru2[0].Stime)
2653     TimeValSub(&rul[1].Utime,&ru2[1].Utime)
2654     TimeValSub(&rul[1].Stime,&ru2[1].Stime)
2655     return rul
2656 }
2657 func TimeValAdd(tv1 syscall.Timeval, tv2 syscall.Timeval)(syscall.Timeval){
2658     tvs := syscall.NsecToTimeval(tv1.Nano() + tv2.Nano())
2659     return tvs
2660 }
2661 /*
2662 func RusageAddv(rul, ru2 [2]syscall.Rusage) ([2]syscall.Rusage){
2663     TimeValAdd(rul[0].Utime,ru2[0].Utime)
2664     TimeValAdd(rul[0].Stime,ru2[0].Stime)
2665     TimeValAdd(rul[1].Utime,ru2[1].Utime)
2666     TimeValAdd(rul[1].Stime,ru2[1].Stime)
2667     return rul
2668 }
2669 */
2670 // <a name="rusage">Resource Usage</a>
2671 func sRusagef(fmtspec string, argv []string, ru [2]syscall.Rusage)(string){
2672     // ru[0] self , ru[1] children
2673     ut := TimeValAdd(ru[0].Utime,ru[1].Utime)
2674     st := TimeValAdd(ru[0].Stime,ru[1].Stime)
2675     uu := (ut.Sec*1000000 + int64(ut.Usec)) * 1000
2676     su := (st.Sec*1000000 + int64(st.Usec)) * 1000
2677     tu := uu + su
2678     ret := fmt.Sprintf("%v/sum",abftime(tu))
2679     ret += fmt.Sprintf(", %v/user",abftime(uu))
2680     ret += fmt.Sprintf(", %v/sys",abftime(su))
2681     return ret
2682 }
2683 }
2684 func Rusagef(fmtspec string, argv []string, ru [2]syscall.Rusage)(string){
2685     ut := TimeValAdd(ru[0].Utime,ru[1].Utime)
2686     st := TimeValAdd(ru[0].Stime,ru[1].Stime)
2687     fmt.Printf("%d.%06ds/u ",ut.Sec,ut.Usec) //ru[1].Utime.Sec,ru[1].Utime.Usec)
2688     fmt.Printf("%d.%06ds/s ",st.Sec,st.Usec) //ru[1].Stime.Sec,ru[1].Stime.Usec)
2689     return ""
2690 }
2691 func Getrusagev()([2]syscall.Rusage){
2692     var ruv = [2]syscall.Rusage{}
2693     syscall.Getrusage(syscall.RUSAGE_SELF,&ruv[0])
2694     syscall.Getrusage(syscall.RUSAGE_CHILDREN,&ruv[1])
2695     return ruv
2696 }
2697 func showRusage(what string,argv []string, ru *syscall.Rusage){
2698     fmt.Printf("%s: ",what);
2699     fmt.Printf("Utr=%d.%06ds",ru.Utime.Sec,ru.Utime.Usec)
2700     fmt.Printf(" Sys=%d.%06ds",ru.Stime.Sec,ru.Stime.Usec)
2701     fmt.Printf(" Rss=%vB",ru.Maxrss)
2702     if isin("-l",argv) {
2703         fmt.Printf(" MinFlt=%v",ru.Minflt)
2704         fmt.Printf(" MajFlt=%v",ru.Majflt)
2705         fmt.Printf(" IxRSS=%vB",ru.Ixrss)
2706         fmt.Printf(" IdRSS=%vB",ru.Idrss)
2707         fmt.Printf(" Nswap=%vB",ru.Nswap)
2708         fmt.Printf(" Read=%v",ru.Inblock)
2709         fmt.Printf(" Write=%v",ru.Oublock)
2710     }
2711     fmt.Printf(" Snd=%v",ru.Msgsnd)
2712     fmt.Printf(" Rcv=%v",ru.Msgrcv)
2713     //if isin("-l",argv) {
2714         fmt.Printf(" Sig=%v",ru.Nsignals)
2715     //}
2716     fmt.Printf("\n");
2717 }
2718 func (gshCtx *GshContext)xTime(argv []string)(bool){
2719     if 2 <= len(argv){
2720         gshCtx.LastRusage = syscall.Rusage{}
2721         rusagev1 := Getrusagev()
2722         fin := gshCtx.gshellv(argv[1:])
2723         rusagev2 := Getrusagev()
2724         showRusage(argv[1],argv,&gshCtx.LastRusage)
2725         rusagev := RusageSubv(rusagev2,rusagev1)
2726         showRusage("self",argv,&rusagev[0])
2727         showRusage("chld",argv,&rusagev[1])
2728         return fin
2729     }else{
2730         rusage:= syscall.Rusage { }
2731         syscall.Getrusage(syscall.RUSAGE_SELF,&rusage)
2732         showRusage("self",argv, &rusage)
2733         syscall.Getrusage(syscall.RUSAGE_CHILDREN,&rusage)
2734         showRusage("chld",argv, &rusage)
2735     }
2736 }
2737 }
2738 func (gshCtx *GshContext)xJobs(argv []string){
2739     fmt.Printf("%d Jobs\n",len(gshCtx.BackGroundJobs))
2740     for ji, pid := range gshCtx.BackGroundJobs {
2741         //wstat := syscall.WaitStatus {0}
2742         rusage := syscall.Rusage { }
2743         //wpid, err := syscall.Wait4(pid,&wstat,syscall.WNOHANG,&rusage);
2744         wpid, err := syscall.Wait4(pid,nil,syscall.WNOHANG,&rusage);
2745         if err != nil {
2746             fmt.Printf("--E-- %%%d [%d] (%v)\n",ji,pid,err)
2747         }else{
2748             fmt.Printf("%%d[%d] (%d)\n",ji,pid,wpid)
2749             showRusage("chld",argv,&rusage)

```

```

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

```

```

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

```



```

3000     }
3001     }
3002     fmt.Printf("%c",fch)
3003     outlen += 1
3004 }
3005 }else{
3006 //fmt.Printf("--D-- print {s}\n")
3007 for i,v := range list {
3008     if 0 < i {
3009         fmt.Printf(div)
3010     }
3011     gsh.printVal(curfmt,v,optv)
3012     outlen += 1
3013 }
3014 }
3015 if 0 < outlen {
3016     fmt.Printf("\n")
3017 }
3018 }
3019 func (gsh*GshContext)Scanv(argv[]string){
3020 //fmt.Printf("--D-- Scnav(%v)\n",argv)
3021 if len(argv) == 1 {
3022     return
3023 }
3024 argv = argv[1:]
3025 fmts := ""
3026 if strBegins(argv[0],"-F") {
3027     fmts = argv[0]
3028     gsh.iDelimiter = fmts
3029     argv = argv[1:]
3030 }
3031 input := strings.Join(argv," ")
3032 if fmts == "" { // simple decomposition
3033     v := scanv(input)
3034     gsh.iValues = v
3035 //fmt.Printf("%v\n",strings.Join(v," "))
3036 }else{
3037     v := make([]string,8)
3038     n,err := fmt.Sscanf(input,fmts,&v[0],&v[1],&v[2],&v[3])
3039     fmt.Printf("--D-- Sscanf ->(%v) n=%d err=(%v)\n",v,n,err)
3040     gsh.iValues = v
3041 }
3042 }
3043 func (gsh*GshContext)Printv(argv[]string){
3044 if false { //@@@
3045     fmt.Printf("%v\n",strings.Join(argv[1:], " "))
3046     return
3047 }
3048 //fmt.Printf("--D-- Printv(%v)\n",argv)
3049 //fmt.Printf("%v\n", strings.Join(gsh.iValues," "))
3050 div := gsh.iDelimiter
3051 fmts := ""
3052 argv = argv[1:]
3053 if 0 < len(argv) {
3054     if strBegins(argv[0],"-F") {
3055         div = argv[0][2:]
3056         argv = argv[1:]
3057     }
3058 }
3059 }
3060 optv := []string{}
3061 for _,v := range argv {
3062     if strBegins(v,"-"){
3063         optv = append(optv,v)
3064         argv = argv[1:]
3065     }else{
3066         break;
3067     }
3068 }
3069 if 0 < len(argv) {
3070     fmts = strings.Join(argv," ")
3071 }
3072 gsh.printf(fmts,div,argv,optv,gsh.iValues)
3073 }
3074 func (gsh*GshContext)Basename(argv[]string){
3075 for i,v := range gsh.iValues {
3076     gsh.iValues[i] = filepath.Base(v)
3077 }
3078 }
3079 func (gsh*GshContext)Sortv(argv[]string){
3080 sv := gsh.iValues
3081 sort.Slice(sv , func(i,j int) bool {
3082     return sv[i] < sv[j]
3083 })
3084 }
3085 func (gsh*GshContext)Shiftv(argv[]string){
3086 vi := len(gsh.iValues)
3087 if 0 < vi {
3088     if isin("-r",argv) {
3089         top := gsh.iValues[0]
3090         gsh.iValues = append(gsh.iValues[1:],top)
3091     }else{
3092         gsh.iValues = gsh.iValues[1:]
3093     }
3094 }
3095 }
3096 }
3097 func (gsh*GshContext)Enq(argv[]string){
3098 }
3099 func (gsh*GshContext)Deq(argv[]string){
3100 }
3101 func (gsh*GshContext)Push(argv[]string){
3102 gsh.iValStack = append(gsh.iValStack,argv[1:])
3103 fmt.Printf("depth=%d\n",len(gsh.iValStack))
3104 }
3105 func (gsh*GshContext)Dump(argv[]string){
3106 for i,v := range gsh.iValStack {
3107     fmt.Printf("%d %v\n",i,v)
3108 }
3109 }
3110 func (gsh*GshContext)Pop(argv[]string){
3111 depth := len(gsh.iValStack)
3112 if 0 < depth {
3113     v := gsh.iValStack[depth-1]
3114     if isin("-cat",argv){
3115         gsh.iValues = append(gsh.iValues,v...)
3116     }else{
3117         gsh.iValues = v
3118     }
3119     gsh.iValStack = gsh.iValStack[0:depth-1]
3120     fmt.Printf("depth=%d %s\n",len(gsh.iValStack),gsh.iValues)
3121 }else{
3122     fmt.Printf("depth=%d\n",depth)
3123 }
3124 }

```

```

3125
3126 // <a name="interpreter">Command Interpreter</a>
3127 func (gshCtx*GshContext)gshellv(argv []string) (fin bool) {
3128     fin = false
3129
3130     if gshCtx.CmdTrace { fmt.Fprintf(os.Stderr,"--I-- gshellv(%d)\n",len(argv)) }
3131     if len(argv) <= 0 {
3132         return false
3133     }
3134     xargv := []string{}
3135     for ai := 0; ai < len(argv); ai++ {
3136         xargv = append(xargv,subst(gshCtx,argv[ai],false))
3137     }
3138     argv = xargv
3139     if false {
3140         for ai := 0; ai < len(argv); ai++ {
3141             fmt.Printf("[%d] %s [%d]%\n",
3142                 ai,argv[ai],len(argv[ai]),argv[ai])
3143         }
3144     }
3145     cmd := argv[0]
3146     if gshCtx.CmdTrace { fmt.Fprintf(os.Stderr,"--I-- gshellv(%d)\%v\n",len(argv),argv) }
3147     switch { // https://tour.golang.org/flowcontrol/11
3148     case cmd == "":
3149         gshCtx.xPwd([]string{}); // empty command
3150     case cmd == "-x":
3151         gshCtx.CmdTrace = ! gshCtx.CmdTrace
3152     case cmd == "-xt":
3153         gshCtx.CmdTime = ! gshCtx.CmdTime
3154     case cmd == "-ot":
3155         gshCtx.sconnect(true, argv)
3156     case cmd == "-ou":
3157         gshCtx.sconnect(false, argv)
3158     case cmd == "-it":
3159         gshCtx.saccept(true, argv)
3160     case cmd == "-iu":
3161         gshCtx.saccept(false, argv)
3162     case cmd == "-i" || cmd == "<" || cmd == "-o" || cmd == ">" || cmd == "-a" || cmd == ">>" || cmd == "-s" || cmd == "><":
3163         gshCtx.redirect(argv)
3164     case cmd == "|":
3165         gshCtx.fromPipe(argv)
3166     case cmd == "args":
3167         gshCtx.Args(argv)
3168     case cmd == "bg" || cmd == "-bg":
3169         rfin := gshCtx.inBackground(argv[1:])
3170         return rfin
3171     case cmd == "-bn":
3172         gshCtx.BaseName(argv)
3173     case cmd == "call":
3174         _ = gshCtx.excommand(false,argv[1:])
3175     case cmd == "cd" || cmd == "ohdir":
3176         gshCtx.xChdir(argv);
3177     case cmd == "-cksum":
3178         gshCtx.xFind(argv)
3179     case cmd == "-sum":
3180         gshCtx.xFind(argv)
3181     case cmd == "-sumtest":
3182         str := ""
3183         if 1 < len(argv) { str = argv[1] }
3184         crc := strCRC32(str,uint64(len(str)))
3185         fprintf(stderr,"%v %v\n",crc,len(str))
3186     case cmd == "close":
3187         gshCtx.xClose(argv)
3188     case cmd == "gcp":
3189         gshCtx.FileCopy(argv)
3190     case cmd == "dec" || cmd == "decode":
3191         gshCtx.Dec(argv)
3192     case cmd == "#define":
3193     case cmd == "dic" || cmd == "d":
3194         xDic(argv)
3195     case cmd == "dump":
3196         gshCtx.Dump(argv)
3197     case cmd == "echo" || cmd == "e":
3198         echo(argv,true)
3199     case cmd == "enc" || cmd == "encode":
3200         gshCtx.Enc(argv)
3201     case cmd == "env":
3202         env(argv)
3203     case cmd == "eval":
3204         xEval(argv[1:],true)
3205     case cmd == "ev" || cmd == "events":
3206         dumpEvents(argv)
3207     case cmd == "exec":
3208         _ = gshCtx.excommand(true,argv[1:])
3209         // should not return here
3210     case cmd == "exit" || cmd == "quit":
3211         // write Result code EXIT to >
3212         return true
3213     case cmd == "fds":
3214         // dump the attributes of fds (of other process)
3215     case cmd == "-find" || cmd == "fin" || cmd == "ufind" || cmd == "uf":
3216         gshCtx.xFind(argv[1:])
3217     case cmd == "fu":
3218         gshCtx.xFind(argv[1:])
3219     case cmd == "fork":
3220         // mainly for a server
3221     case cmd == "-gen":
3222         gshCtx.gen(argv)
3223     case cmd == "-go":
3224         gshCtx.xGo(argv)
3225     case cmd == "-grep":
3226         gshCtx.xFind(argv)
3227     case cmd == "gdeg":
3228         gshCtx.Deg(argv)
3229     case cmd == "genq":
3230         gshCtx.Enq(argv)
3231     case cmd == "gpop":
3232         gshCtx.Pop(argv)
3233     case cmd == "gpush":
3234         gshCtx.Push(argv)
3235     case cmd == "history" || cmd == "hi": // hi should be alias
3236         gshCtx.xHistory(argv)
3237     case cmd == "jobs":
3238         gshCtx.xJobs(argv)
3239     case cmd == "lnsp" || cmd == "nls":
3240         gshCtx.SplitLine(argv)
3241     case cmd == "-ls":
3242         gshCtx.xFind(argv)
3243     case cmd == "nop":
3244         // do nothing
3245     case cmd == "pipe":
3246         gshCtx.xOpen(argv)
3247     case cmd == "plug" || cmd == "plugin" || cmd == "pin":
3248         gshCtx.xPlugin(argv[1:])
3249     case cmd == "print" || cmd == "-pr":

```

```

3250 // output internal slice // also sprintf should be
3251 gshCtx.Printf(argv)
3252 case cmd == "ps":
3253 gshCtx.XPs(argv)
3254 case cmd == "pstitle":
3255 // to be gsh.title
3256 case cmd == "rexeod" || cmd == "rexd":
3257 gshCtx.RexeServer(argv)
3258 case cmd == "rexeoc" || cmd == "rex":
3259 gshCtx.RexeClient(argv)
3260 case cmd == "repeat" || cmd == "rep": // repeat cond command
3261 gshCtx.repeat(argv)
3262 case cmd == "replay":
3263 gshCtx.xReplay(argv)
3264 case cmd == "scan":
3265 // scan input (or so in fscanf) to internal slice (like Files or map)
3266 gshCtx.Scanv(argv)
3267 case cmd == "set":
3268 // set name ...
3269 case cmd == "serv":
3270 gshCtx.httpServer(argv)
3271 case cmd == "shift":
3272 gshCtx.Shiftv(argv)
3273 case cmd == "sleep":
3274 gshCtx.sleep(argv)
3275 case cmd == "-sort":
3276 gshCtx.Sortv(argv)
3277
3278 case cmd == "j" || cmd == "join":
3279 gshCtx.Rjoin(argv)
3280 case cmd == "a" || cmd == "alpa":
3281 gshCtx.Rexec(argv)
3282 case cmd == "jcd" || cmd == "jchdir":
3283 gshCtx.Rchdir(argv)
3284 case cmd == "jget":
3285 gshCtx.Rget(argv)
3286 case cmd == "jls":
3287 gshCtx.Rls(argv)
3288 case cmd == "jput":
3289 gshCtx.Rput(argv)
3290 case cmd == "jpwd":
3291 gshCtx.Rpwd(argv)
3292
3293 case cmd == "time":
3294 fin = gshCtx.xTime(argv)
3295 case cmd == "ungets":
3296 if l < len(argv) {
3297 ungets(argv[l]+\n")
3298 }else{
3299 }
3300 case cmd == "pwd":
3301 gshCtx.xPwd(argv);
3302 case cmd == "ver" || cmd == "-ver" || cmd == "version":
3303 gshCtx.showVersion(argv)
3304 case cmd == "where":
3305 // data file or so?
3306 case cmd == "which":
3307 which("PATH",argv);
3308 default:
3309 if gshCtx.whichPlugin(cmd,[]string{"-s"}) != nil {
3310 gshCtx.xPlugin(argv)
3311 }else{
3312 notfound,_ := gshCtx.excommand(false,argv)
3313 if notfound {
3314 fmt.Printf("--E-- command not found (%v)\n",cmd)
3315 }
3316 }
3317 }
3318 return fin
3319 }
3320
3321 func (gsh*GshContext)gshell(gline string) (rfin bool) {
3322 argv := strings.Split(string(gline)," ")
3323 fin := gsh.gshellv(argv)
3324 return fin
3325 }
3326 func (gsh*GshContext)tgshell(gline string)(xfin bool){
3327 start := time.Now()
3328 fin := gsh.gshell(gline)
3329 end := time.Now()
3330 elps := end.Sub(start);
3331 if gsh.CmdTime {
3332 fmt.Printf("--T-- " + time.Now().Format(time.Stamp) + " (%d.%09ds)\n",
3333 elps/1000000000,elps%1000000000)
3334 }
3335 return fin
3336 }
3337 func Ttyid() (int) {
3338 fi, err := os.Stdin.Stat()
3339 if err != nil {
3340 return 0;
3341 }
3342 //fmt.Printf("Stdin: %v Dev=%d\n",
3343 // fi.Mode(),fi.Mode()&os.ModeDevice)
3344 if (fi.Mode() & os.ModeDevice) != 0 {
3345 stat := syscall.Stat_t{};
3346 err := syscall.Fstat(0,&stat)
3347 if err != nil {
3348 //fmt.Printf("--I-- Stdin: (%v)\n",err)
3349 }else{
3350 //fmt.Printf("--I-- Stdin: rdev=%d %d\n",
3351 // stat.Rdev&0xFF,stat.Rdev);
3352 //fmt.Printf("--I-- Stdin: tty%d\n",stat.Rdev&0xFF);
3353 return int(stat.Rdev & 0xFF)
3354 }
3355 }
3356 return 0
3357 }
3358 func (gshCtx *GshContext) ttyfile() string {
3359 //fmt.Printf("--I-- GSH_HOME=%s\n",gshCtx.GshHomeDir)
3360 ttyfile := gshCtx.GshHomeDir + "/" + "gsh-tty" +
3361 fmt.Sprintf("%02d",gshCtx.TerminalId)
3362 //strconv.Itoa(gshCtx.TerminalId)
3363 //fmt.Printf("--I-- ttyfile=%s\n",ttyfile)
3364 return ttyfile
3365 }
3366 func (gshCtx *GshContext) ttyline>(*os.File){
3367 file, err := os.OpenFile(gshCtx.ttyfile(),os.O_RDWR|os.O_CREATE|os.O_TRUNC,0600)
3368 if err != nil {
3369 fmt.Printf("--F-- cannot open %s (%s)\n",gshCtx.ttyfile(),err)
3370 return file;
3371 }
3372 return file
3373 }
3374 func (gshCtx *GshContext)getline(hix int, skipping bool, prevline string) (string) {

```

```

3375 if( skipping ){
3376     reader := bufio.NewReaderSize(os.Stdin,LINESIZE)
3377     line, _, _ := reader.ReadLine()
3378     return string(line)
3379 }else
3380 if true {
3381     return xgetline(hix,prevline,gshCtx)
3382 }
3383 /*
3384 else
3385 if( with_exgetline && gshCtx.GetLine != "" ){
3386     //var xhix int64 = int64(hix); // cast
3387     newenv := os.Environ()
3388     newenv = append(newenv, "GSH_LINENO="+strconv.FormatInt(int64(hix),10) )
3389
3390     tty := gshCtx.ttyline()
3391     tty.WriteString(prevline)
3392     Pa := os.ProcAttr {
3393         "", // start dir
3394         newenv, //os.Environ(),
3395         []*os.File{os.Stdin,os.Stdout,os.Stderr,tty},
3396         nil,
3397     }
3398     //fmt.Printf("--I-- getline=%s // %s\n",gsh_getlinev[0],gshCtx.GetLine)
3399     proc, err := os.StartProcess(gsh_getlinev[0],[string{"getline","getline"},&Pa)
3400     if err != nil {
3401         fmt.Printf("--F-- getline process error (%v)\n",err)
3402         // for ; ; { }
3403         return "exit (getline program failed)"
3404     }
3405     //stat, err := proc.Wait()
3406     proc.Wait()
3407     buff := make([]byte,LINESIZE)
3408     count, err := tty.Read(buff)
3409     //_, err = tty.Read(buff)
3410     //fmt.Printf("--D-- getline (%d)\n",count)
3411     if err != nil {
3412         if ! (count == 0) { // && err.String() == "EOF" } {
3413             fmt.Printf("--E-- getline error (%s)\n",err)
3414         }
3415     }else{
3416         //fmt.Printf("--I-- getline OK \"%s\"\n",buff)
3417     }
3418     tty.Close()
3419     gline := string(buff[0:count])
3420     return gline
3421 }else
3422 */
3423 {
3424     // if isatty {
3425     fmt.Printf("!%d",hix)
3426     fmt.Print(PROMPT)
3427     // }
3428     reader := bufio.NewReaderSize(os.Stdin,LINESIZE)
3429     line, _, _ := reader.ReadLine()
3430     return string(line)
3431 }
3432 }
3433
3434 //== begin ===== getline
3435 /*
3436 * getline.c
3437 * 2020-0819 extracted from dog.c
3438 * getline.go
3439 * 2020-0822 ported to Go
3440 */
3441 /*
3442 package main // getline main
3443 import (
3444     "fmt" // <a href="https://golang.org/pkg/fmt/">fmt</a>
3445     "strings" // <a href="https://golang.org/pkg/strings/">strings</a>
3446     "os" // <a href="https://golang.org/pkg/os/">os</a>
3447     "syscall" // <a href="https://golang.org/pkg/syscall/">syscall</a>
3448     //"bytes" // <a href="https://golang.org/pkg/bytes/">bytes</a>
3449     //"os/exec" // <a href="https://golang.org/pkg/os/exec/">os/exec</a>
3450 )
3451 */
3452
3453 // C language compatibility functions
3454 var errno = 0
3455 var stdin *os.File = os.Stdin
3456 var stdout *os.File = os.Stdout
3457 var stderr *os.File = os.Stderr
3458 var EOF = -1
3459 var NULL = 0
3460 type FILE os.File
3461 type StrBuff []byte
3462 var NULL_FP *os.File = nil
3463 var NULLSP = 0
3464 //var LINESIZE = 1024
3465
3466 func system(cmdstr string)(int){
3467     PA := syscall.ProcAttr {
3468         "", // the starting directory
3469         os.Environ(),
3470         []uintptr{os.Stdin.Fd(),os.Stdout.Fd(),os.Stderr.Fd()},
3471         nil,
3472     }
3473     argv := strings.Split(cmdstr, " ")
3474     pid,err := syscall.ForkExec(argv[0],argv,&PA)
3475     if( err != nil ){
3476         fmt.Printf("--E-- syscall(%v) err(%v)\n",cmdstr,err)
3477     }
3478     syscall.Wait4(pid,nil,0,nil)
3479
3480     /*
3481     argv := strings.Split(cmdstr, " ")
3482     fmt.Fprintf(os.Stderr,"--I-- system(%v)\n",argv)
3483     //cmd := exec.Command(argv[0:]...)
3484     cmd := exec.Command(argv[0],argv[1],argv[2])
3485     cmd.Stdin = strings.NewReader("output of system")
3486     var out bytes.Buffer
3487     cmd.Stdout = &out
3488     var serr bytes.Buffer
3489     cmd.Stderr = &serr
3490     err := cmd.Run()
3491     if err != nil {
3492         fmt.Fprintf(os.Stderr,"--E-- system(%v)err(%v)\n",argv,err)
3493         fmt.Printf("ERR:%s\n",serr.String())
3494     }else{
3495         fmt.Printf("%s",out.String())
3496     }
3497     */
3498     return 0
3499 }

```

```

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

```

```

3625     fp.Write(buf)
3626     return 0
3627 }
3628 func xfputss(str string, fp*os.File)(int){
3629     return fputs([]byte(str),fp)
3630 }
3631 func sscanf(str StrBuff,fmts string, params ...interface{})(int){
3632     fmt.Sscanf(string(str[0:strlen(str)]),fmts,params...)
3633     return 0
3634 }
3635 func fprintf(fp*os.File,fmts string, params ...interface{})(int){
3636     fmt.Fprintf(fp,fmts,params...)
3637     return 0
3638 }
3639 }
3640 // <a name="IME">Command Line IME</a>
3641 //----- MyIME
3642 var MyIMEVER = "MyIME/0.0.2";
3643 type RomKana struct {
3644     dic string // dictionary ID
3645     pat string // input pattern
3646     out string // output pattern
3647     hit int64 // count of hit and used
3648 }
3649 var dicents = 0
3650 var romkana [1024]RomKana
3651 var Romkan []RomKana
3652 }
3653 func isinDic(str string)(int){
3654     for i,v := range Romkan {
3655         if v.pat == str {
3656             return i
3657         }
3658     }
3659     return -1
3660 }
3661 const (
3662     DIC_COM_LOAD = "im"
3663     DIC_COM_DUMP = "s"
3664     DIC_COM_LIST = "ls"
3665     DIC_COM_ENA = "en"
3666     DIC_COM_DIS = "di"
3667 )
3668 func helpDic(argv []string){
3669     out := stderr
3670     cmd := ""
3671     if 0 < len(argv) { cmd = argv[0] }
3672     fprintf(out,"--- %v Usage\n",cmd)
3673     fprintf(out,"... Commands\n")
3674     fprintf(out,"... %v %v [dicName] [dicURL ] -- Import dictionary\n",cmd,DIC_COM_LOAD)
3675     fprintf(out,"... %v %v [pattern] -- Search in dictionary\n",cmd,DIC_COM_DUMP)
3676     fprintf(out,"... %v %v [dicName] -- List dictionaries\n",cmd,DIC_COM_LIST)
3677     fprintf(out,"... %v %v [dicName] -- Disable dictionaries\n",cmd,DIC_COM_DIS)
3678     fprintf(out,"... %v %v [dicName] -- Enable dictionaries\n",cmd,DIC_COM_ENA)
3679     fprintf(out,"... Keys .. %v\n","ESC can be used for '\\')")
3680     fprintf(out,"... \\c -- Reverse the case of the last character\n",)
3681     fprintf(out,"... \\i -- Replace input with translated text\n",)
3682     fprintf(out,"... \\j -- On/Off translation mode\n",)
3683     fprintf(out,"... \\l -- Force Lower Case\n",)
3684     fprintf(out,"... \\u -- Force Upper Case (software CapsLock)\n",)
3685     fprintf(out,"... \\v -- Show translation actions\n",)
3686     fprintf(out,"... \\x -- Replace the last input character with it Hexa-Decimal\n",)
3687 }
3688 func xDic(argv[]string){
3689     if len(argv) <= 1 {
3690         helpDic(argv)
3691         return
3692     }
3693     argv = argv[1:]
3694     var debug = false
3695     var info = false
3696     var silent = false
3697     var dump = false
3698     var builtin = false
3699     cmd := argv[0]
3700     argv = argv[1:]
3701     opt := ""
3702     arg := ""
3703 }
3704 if 0 < len(argv) {
3705     arg1 := argv[0]
3706     if arg1[0] == '-' {
3707         switch arg1 {
3708             default: fmt.Printf("--Ed-- Unknown option(%v)\n",arg1)
3709                 return
3710             case "-b": builtin = true
3711             case "-d": debug = true
3712             case "-s": silent = true
3713             case "-v": info = true
3714         }
3715     }
3716     opt = arg1
3717     argv = argv[1:]
3718 }
3719 }
3720 dicName := ""
3721 dicURL := ""
3722 if 0 < len(argv) {
3723     arg = argv[0]
3724     dicName = arg
3725     argv = argv[1:]
3726 }
3727 if 0 < len(argv) {
3728     dicURL = argv[0]
3729     argv = argv[1:]
3730 }
3731 if false {
3732     fprintf(stderr,"--Dd-- com(%v) opt(%v) arg(%v)\n",cmd,opt,arg)
3733 }
3734 if cmd == DIC_COM_LOAD {
3735     //dicType := ""
3736     dicBody := ""
3737     if !builtin && dicName != "" && dicURL == "" {
3738         f,err := os.Open(dicName)
3739         if err == nil {
3740             dicURL = dicName
3741         }else{
3742             f,err = os.Open(dicName+".html")
3743             if err == nil {
3744                 dicURL = dicName+".html"
3745             }else{
3746                 f,err = os.Open("gshdic-"+dicName+".html")
3747                 if err == nil {
3748                     dicURL = "gshdic-"+dicName+".html"
3749                 }
3750             }
3751         }
3752     }
3753 }

```

```

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

```

```

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

```



```

4000 const (
4001     EV_MODE   = 255
4002     EV_IDLE   = 254
4003     EV_TIMEOUT = 253
4004
4005     GO_UP     = 252 // k
4006     GO_DOWN   = 251 // j
4007     GO_RIGHT  = 250 // l
4008     GO_LEFT   = 249 // h
4009     DEL_RIGHT = 248 // x
4010     GO_TOPL   = 'A'-0x40 // 0
4011     GO_ENDL   = 'E'-0x40 // $
4012
4013     GO_TOPW   = 239 // b
4014     GO_ENDW   = 238 // e
4015     GO_NEXTW  = 237 // w
4016
4017     GO_FORWCH = 229 // f
4018     GO_PAIRCH = 228 // %
4019
4020     GO_DEL    = 219 // d
4021
4022     HI_SRCH_FW = 209 // /
4023     HI_SRCH_BK = 208 // ?
4024     HI_SRCH_RFW = 207 // n
4025     HI_SRCH_RBK = 206 // N
4026 )
4027
4028 // should return number of octets ready to be read immediately
4029 //fprintf(stderr, "\n--Select(%v %v)\n", err, r.Bits[0])
4030
4031
4032 var EventRecvFd = -1 // file descriptor
4033 var EventSendFd = -1
4034 const EventFdOffset = 1000000
4035 const NormalFdOffset = 100
4036
4037 func putEvent(event int, evarg int){
4038     if true {
4039         if EventRecvFd < 0 {
4040             var pv = []int{-1,-1}
4041             syscall.Pipe(pv)
4042             EventRecvFd = pv[0]
4043             EventSendFd = pv[1]
4044             //fmt.Printf("--De-- EventPipe created[%v,%v]\n", EventRecvFd, EventSendFd)
4045         }
4046     }else{
4047         if EventRecvFd < 0 {
4048             // the document differs from this spec
4049             // https://golang.org/src/syscall/syscall_unix.go?s=8096:8159#L1340
4050             sv, err := syscall.Socketpair(syscall.AF_UNIX, syscall.SOCK_STREAM, 0)
4051             EventRecvFd = sv[0]
4052             EventSendFd = sv[1]
4053             if err != nil {
4054                 fmt.Printf("--De-- EventSock created[%v,%v]({%v})\n",
4055                     EventRecvFd, EventSendFd, err)
4056             }
4057         }
4058     }
4059     var buf = []byte{ byte(event) }
4060     n, err := syscall.Write(EventSendFd, buf)
4061     if err != nil {
4062         fmt.Printf("--De-- putEvent[%v]({%3v}) (%v %v)\n", EventSendFd, event, n, err)
4063     }
4064 }
4065 func ungets(str string){
4066     for _, ch := range str {
4067         putEvent(int(ch), 0)
4068     }
4069 }
4070 func (gsh*GshContext)xReplay(argv []string){
4071     hix := 0
4072     tempo := 1.0
4073     xtempo := 1.0
4074     repeat := 1
4075
4076     for _, a := range argv { // tempo
4077         if strBegins(a, "x") {
4078             fmt.Sscanf(a[1:], "%f", &xtempo)
4079             tempo = 1 / xtempo
4080             //fprintf(stderr, "--Dr-- tempo=[%v]%v\n", a[2:], tempo);
4081         }else
4082         if strBegins(a, "r") { // repeat
4083             fmt.Sscanf(a[1:], "%v", &repeat)
4084         }else
4085         if strBegins(a, "l") {
4086             fmt.Sscanf(a[1:], "%d", &hix)
4087         }else{
4088             fmt.Sscanf(a, "%d", &hix)
4089         }
4090     }
4091     if hix == 0 || len(argv) <= 1 {
4092         hix = len(gsh.CommandHistory)-1
4093     }
4094     fmt.Printf("--Ir-- Replay(!%v x%v r%v)\n", hix, xtempo, repeat)
4095     //dumpEvents(hix)
4096     //gsh.xScanReplay(hix, false, repeat, tempo, argv)
4097     go gsh.xScanReplay(hix, true, repeat, tempo, argv)
4098 }
4099
4100 // <a href="https://golang.org/pkg/syscall/#FdSet">syscall.Select</a>
4101 // 2020-0827 GShell-0.2.3
4102 /*
4103 func FpollIn1(fp *os.File, usec int) (uintptr){
4104     nfd := 1
4105
4106     rdv := syscall.FdSet {}
4107     fd1 := fp.Fd()
4108     bank1 := fd1/32
4109     mask1 := int32(1 << fd1)
4110     rdv.Bits[bank1] = mask1
4111
4112     fd2 := -1
4113     bank2 := -1
4114     var mask2 int32 = 0
4115
4116     if 0 <= EventRecvFd {
4117         fd2 = EventRecvFd
4118         nfd = fd2 + 1
4119         bank2 = fd2/32
4120         mask2 = int32(1 << fd2)
4121         rdv.Bits[bank2] |= mask2
4122         //fmt.Printf("--De-- EventPoll mask added [%d][%v][%v]\n", fd2, bank2, mask2)
4123     }
4124 }

```

```

4125 tout := syscall.NsecToTimeval(int64(usec*1000))
4126 //n, err := syscall.Select(nfd, &rdv, nil, nil, &tout) // spec. mismatch
4127 err := syscall.Select(nfd, &rdv, nil, nil, &tout)
4128 if err != nil {
4129     //fmt.Printf("--De-- select() err(%v)\n", err)
4130 }
4131 if err == nil {
4132     if 0 <= fd2 && (rdv.Bits[bank2] & mask2) != 0 {
4133         if false {
4134             fmt.Printf("--De-- got Event\n")
4135         }
4136         return uintptr(EventFdOffset + fd2)
4137     }else
4138     if (rdv.Bits[bank1] & mask1) != 0 {
4139         return uintptr(NormalFdOffset + fd1)
4140     }else{
4141         return 1
4142     }
4143 }else{
4144     return 0
4145 }
4146 }
4147 */
4148 func fgetcTimeout1(fp *os.File, usec int)(int){
4149     READ1:
4150     //readyFd := FpollIn1(fp, usec)
4151     readyFd := CfpollIn1(fp, usec)
4152     if readyFd < 100 {
4153         return EV_TIMEOUT
4154     }
4155 }
4156 var buf [1]byte
4157
4158 if EventFdOffset <= readyFd {
4159     fd := int(readyFd - EventFdOffset)
4160     _, err := syscall.Read(fd, buf[0:1])
4161     if( err != nil ){
4162         return EOF;
4163     }else{
4164         if buf[0] == EV_MODE {
4165             recvEvent(fd)
4166             goto READ1
4167         }
4168         return int(buf[0])
4169     }
4170 }
4171
4172 _, err := fp.Read(buf[0:1])
4173 if( err != nil ){
4174     return EOF;
4175 }else{
4176     return int(buf[0])
4177 }
4178 }
4179
4180 func visibleChar(ch int)(string){
4181     switch {
4182     case '!' <= ch && ch <= '-':
4183         return string(ch)
4184     }
4185     switch ch {
4186     case '\t': return "\\s"
4187     case '\n': return "\\n"
4188     case '\r': return "\\r"
4189     case '\t': return "\\t"
4190     }
4191     switch ch {
4192     case 0x00: return "NUL"
4193     case 0x07: return "BEL"
4194     case 0x08: return "BS"
4195     case 0x0E: return "SO"
4196     case 0x0F: return "SI"
4197     case 0x1B: return "ESC"
4198     case 0x7F: return "DEL"
4199     }
4200     switch ch {
4201     case EV_IDLE: return fmt.Sprintf("IDLE")
4202     case EV_MODE: return fmt.Sprintf("MODE")
4203     }
4204     return fmt.Sprintf("%X", ch)
4205 }
4206 func recvEvent(fd int){
4207     var buf = make([]byte, 1)
4208     _, err := syscall.Read(fd, buf[0:1])
4209     if( buf[0] != 0 ){
4210         romkanmode = true
4211     }else{
4212         romkanmode = false
4213     }
4214 }
4215 func (gsh*GshContext)xScanReplay(hix int, replay bool, repeat int, tempo float64, argv []string){
4216     var Start time.Time
4217     var events = []Event{}
4218     for _, e := range Events {
4219         if hix == 0 || e.CmdIndex == hix {
4220             events = append(events, e)
4221         }
4222     }
4223     elen := len(events)
4224     if 0 < elen {
4225         if events[elen-1].event == EV_IDLE {
4226             events = events[0:elen-1]
4227         }
4228     }
4229     for r := 0; r < repeat; r++ {
4230         for i, e := range events {
4231             nano := e.when.Nanosecond()
4232             micro := nano / 1000
4233             if Start.Second() == 0 {
4234                 Start = time.Now()
4235             }
4236             diff := time.Now().Sub(Start)
4237             if replay {
4238                 if e.event != EV_IDLE {
4239                     putEvent(e.event, 0)
4240                     if e.event == EV_MODE { // event with arg
4241                         putEvent(int(e.evarg), 0)
4242                     }
4243                 }
4244             }else{
4245                 fmt.Printf("%7.3fms %#-3v !%-3v [%v.%06d] %3v %02X %-4v %10.3fms\n",
4246                     float64(diff)/1000000.0,
4247                     i,
4248                     e.CmdIndex,
4249                     e.when.Format(time.Stamp), micro,

```

```

4250         e.event,e.event,visibleChar(e.event),
4251         float64(e.evarg)/1000000.0)
4252     }
4253     if e.event == EV_IDLE {
4254         d := time.Duration(float64(time.Duration(e.evarg)) * tempo)
4255         //nsleep(time.Duration(e.evarg))
4256         nsleep(d)
4257     }
4258 }
4259 }
4260 }
4261 func dumpEvents(arg []string){
4262     hix := 0
4263     if 1 < len(arg) {
4264         fmt.Sprintf(arg[1], "%d", &hix)
4265     }
4266     for i,e := range Events {
4267         nano := e.when.Nanosecond()
4268         micro := nano / 1000
4269         //if e.event != EV_TIMEOUT {
4270         if hix == 0 || e.CmdIndex == hix {
4271             fmt.Printf("#%-3v !%-3v [%v.%06d] %3v %02X %-4v %10.3fms\n",i,
4272             e.CmdIndex,
4273             e.when.Format(time.Stamp),micro,
4274             e.event,e.event,visibleChar(e.event),float64(e.evarg)/1000000.0)
4275         }
4276         ( //)
4277     }
4278 }
4279 func fgetcTimeout(fp *os.File,usec int)(int){
4280     ch := fgetcTimeout1(fp,usec)
4281     if ch != EV_TIMEOUT {
4282         now := Time.Now()
4283         if 0 < len(Events) {
4284             last := Events[len(Events)-1]
4285             dura := int64(now.Sub(last.when))
4286             Events = append(Events,Event{last.when,EV_IDLE,dura,last.CmdIndex})
4287         }
4288         Events = append(Events,Event{time.Now(),ch,0,CmdIndex})
4289     }
4290     return ch
4291 }
4292 }
4293 var TtyMaxCol = 72 // to be obtained by ioctl?
4294 var EscTimeout = (100*1000)
4295 var (
4296     MODE_VicMode    bool    // vi compatible command mode
4297     MODE_ShowMode   bool
4298     romkanmode     bool    // shown translation mode, the mode to be retained
4299     MODE_Recursive  bool    // recursive translation
4300     MODE_CapsLock   bool    // software CapsLock
4301     MODE_LowerLock  bool    // force lower-case character lock
4302     MODE_Viinsert  int     // visible insert mode, should be like "I" icon in X Window
4303     MODE_ViTrace   bool    // output newline before translation
4304 )
4305 type IInput struct {
4306     lno      int
4307     lastlno int
4308     pch      []int // input queue
4309     prompt   string
4310     line     string
4311     right    string
4312     inJmode  bool
4313     pinJmode bool
4314     waitingMeta string // waiting meta character
4315     LastCmd   string
4316 }
4317 func (iin*IInput)Getc(timeoutUs int)(int){
4318     ch1 := EOF
4319     ch2 := EOF
4320     ch3 := EOF
4321     if( 0 < len(iin.pch) ){ // deQ
4322         ch1 = iin.pch[0]
4323         iin.pch = iin.pch[1:]
4324     }else{
4325         ch1 = fgetcTimeout(stdin,timeoutUs);
4326     }
4327     if( ch1 == 033 ){ // escape sequence
4328         ch2 = fgetcTimeout(stdin,EscTimeout);
4329         if( ch2 == EV_TIMEOUT ){
4330             }else{
4331                 ch3 = fgetcTimeout(stdin,EscTimeout);
4332                 if( ch3 == EV_TIMEOUT ){
4333                     iin.pch = append(iin.pch,ch2) // enQ
4334                 }else{
4335                     switch( ch2 ){
4336                     default:
4337                         iin.pch = append(iin.pch,ch2) // enQ
4338                         iin.pch = append(iin.pch,ch3) // enQ
4339                     case '[':
4340                         switch( ch3 ){
4341                         case 'A': ch1 = GO_UP; // ^
4342                         case 'B': ch1 = GO_DOWN; // v
4343                         case 'C': ch1 = GO_RIGHT; // >
4344                         case 'D': ch1 = GO_LEFT; // <
4345                         case '3':
4346                             ch4 := fgetcTimeout(stdin,EscTimeout);
4347                             if( ch4 == '-' ){
4348                                 //fprintf(stderr, "x[%02X %02X %02X %02X]\n",ch1,ch2,ch3,ch4);
4349                                 ch1 = DEL_RIGHT
4350                             }
4351                         }
4352                     case '\\':
4353                         //ch4 := fgetcTimeout(stdin,EscTimeout);
4354                         //fprintf(stderr, "y[%02X %02X %02X %02X]\n",ch1,ch2,ch3,ch4);
4355                         switch( ch3 ){
4356                         case '-': ch1 = DEL_RIGHT
4357                         }
4358                     }
4359                 }
4360             }
4361         }
4362         return ch1
4363     }
4364 }
4365 func (inn*IInput)clearline(){
4366     var i int
4367     fprintf(stderr, "\r");
4368     // should be ANSI ESC sequence
4369     for i = 0; i < TtyMaxCol; i++ { // to the max. position in this input action
4370         fputc(' ',os.Stderr);
4371     }
4372     fprintf(stderr, "\r");
4373 }
4374 func (iin*IInput)Redraw(){
4375     redraw(iin,iin.lno,iin.line,iin.right)

```

```

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

```

```

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

```

```

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

```

```

4750     }else{
4751         return false
4752     }
4753 }
4754 // 0.2.8 2020-0902 created
4755 func (iin*IInput)SearchBackward(pat string)(bool){
4756     line := iin.line
4757     found := -1
4758     i := len(line)-1
4759     for i = i; 0 <= i; i-- {
4760         _,z := utf8.DecodeRuneInString(line[i:])
4761         if z <= 0 {
4762             continue
4763         }
4764         //fprintf(stderr,"-- %v %v\n",pat,line[i:])
4765         if strBegins(line[i:],pat) {
4766             found = i
4767             break
4768         }
4769     }
4770     //fprintf(stderr,"--%d\n",found)
4771     if 0 <= found {
4772         iin.right = line[found:] + iin.right
4773         iin.line = line[0:found]
4774         return true
4775     }else{
4776         return false
4777     }
4778 }
4779 // 0.2.8 2020-0902 created
4780 // search from top, end, or current position
4781 func (gsh*GshContext)SearchHistory(pat string, forw bool)(bool,string){
4782     if forw {
4783         for _,v := range gsh.CommandHistory {
4784             if 0 <= strings.Index(v.CmdLine,pat) {
4785                 //fprintf(stderr,"\n--De-- found !%v [%v]%v\n",i,pat,v.CmdLine)
4786                 return true,v.CmdLine
4787             }
4788         }
4789     }else{
4790         hlen := len(gsh.CommandHistory)
4791         for i := hlen-1; 0 <= i; i-- {
4792             v := gsh.CommandHistory[i]
4793             if 0 <= strings.Index(v.CmdLine,pat) {
4794                 //fprintf(stderr,"\n--De-- found !%v [%v]%v\n",i,pat,v.CmdLine)
4795                 return true,v.CmdLine
4796             }
4797         }
4798     }
4799     //fprintf(stderr,"\n--De-- not-found(%v)\n",pat)
4800     return false,"(Not Found in History)"
4801 }
4802 // 0.2.8 2020-0902 created
4803 func (iin*IInput)GotoFORWSTR(pat string,gsh*GshContext){
4804     found := false
4805     if 0 <= len(iin.right) {
4806         found = iin.SearchForward(pat)
4807     }
4808     if !found {
4809         found,line := gsh.SearchHistory(pat,true)
4810         if found {
4811             iin.line = line
4812             iin.right = ""
4813         }
4814     }
4815 }
4816 func (iin*IInput)GotoBACKSTR(pat string,gsh*GshContext){
4817     found := false
4818     if 0 <= len(iin.line) {
4819         found = iin.SearchBackward(pat)
4820     }
4821     if !found {
4822         found,line := gsh.SearchHistory(pat,false)
4823         if found {
4824             iin.line = line
4825             iin.right = ""
4826         }
4827     }
4828 }
4829 func (iin*IInput)getString1(prompt string)(string){ // should be editable
4830     iin.clearline();
4831     fprintf(stderr,"\r%v",prompt)
4832     str := ""
4833     for {
4834         ch := iin.Getc(10*1000*1000)
4835         if ch == '\n' || ch == '\r' {
4836             break
4837         }
4838         sch := string(ch)
4839         str += sch
4840         fprintf(stderr,"%s",sch)
4841     }
4842     return str
4843 }
4844
4845 // search pattern must be an array and selectable with ^N/^P
4846 var SearchPat = ""
4847 var SearchForw = true
4848
4849 func (iin*IInput)xgetline1(prevline string,gsh*GshContext)(string){
4850     var ch int;
4851
4852     MODE_ShowMode = false
4853     MODE_VicMode = false
4854     iin.Redraw();
4855     first := true
4856
4857     for cix := 0; ; cix++ {
4858         iin.pinJmode = iin.inJmode
4859         iin.inJmode = false
4860
4861         ch = iin.Getc(1000*1000)
4862
4863         if ch != EV_TIMEOUT && first {
4864             first = false
4865             mode := 0
4866             if romkanmode {
4867                 mode = 1
4868             }
4869             now := time.Now()
4870             Events = append(Events,Event{now,EV_MODE,int64(mode),CmdIndex})
4871         }
4872         if ch == 033 {
4873             MODE_ShowMode = true
4874             MODE_VicMode = !MODE_VicMode

```

```

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

```



```

5000     }else
5001     /*
5002     if( ch == 'k' || ch == 'K' ){
5003         MODE_Recursive = IMODE_Recursive
5004         iin.Translate(cmdch);
5005         continue
5006     }else
5007     if( ch == 'j' || ch == 'J' ){
5008         iin.Translate(cmdch);
5009         continue
5010     }else
5011     if( ch == 'i' || ch == 'I' ){
5012         iin.Replace(cmdch);
5013         continue
5014     }else
5015     if( ch == 'l' || ch == 'L' ){
5016         MODE_LowerLock = IMODE_LowerLock
5017         MODE_CapsLock = false
5018         if MODE_ViTrace {
5019             fprintf(stderr, "%v\r\n", string(cmdch));
5020         }
5021         iin.Redraw();
5022         continue
5023     }else
5024     if( ch == 'u' || ch == 'U' ){
5025         MODE_CapsLock = IMODE_CapsLock
5026         MODE_LowerLock = false
5027         if MODE_ViTrace {
5028             fprintf(stderr, "%v\r\n", string(cmdch));
5029         }
5030         iin.Redraw();
5031         continue
5032     }else
5033     if( ch == 'v' || ch == 'V' ){
5034         MODE_ViTrace = IMODE_ViTrace
5035         if MODE_ViTrace {
5036             fprintf(stderr, "%v\r\n", string(cmdch));
5037         }
5038         iin.Redraw();
5039         continue
5040     }else
5041     if( ch == 'c' || ch == 'C' ){
5042         if 0 < len(iin.line) {
5043             xline,tail := delTailChar(iin.line)
5044             if len([]byte(tail)) == 1 {
5045                 ch = int(tail[0])
5046                 if( 'a' <= ch && ch <= 'z' ){
5047                     ch = ch + 'A'-'a'
5048                 }else
5049                 if( 'A' <= ch && ch <= 'Z' ){
5050                     ch = ch + 'a'-'A'
5051                 }
5052                 iin.line = xline + string(ch)
5053             }
5054         }
5055         if MODE_ViTrace {
5056             fprintf(stderr, "%v\r\n", string(cmdch));
5057         }
5058         iin.Redraw();
5059         continue
5060     }else{
5061         iin.pch = append(iin.pch,ch) // push
5062         ch = '\\'
5063     }
5064 }
5065 switch( ch ){
5066 case 'P'-0x40: ch = GO_UP
5067 case 'N'-0x40: ch = GO_DOWN
5068 case 'B'-0x40: ch = GO_LEFT
5069 case 'F'-0x40: ch = GO_RIGHT
5070 }
5071 //fprintf(stderr, "B[802X]\n",ch);
5072 switch( ch ){
5073 case 0:
5074     continue;
5075
5076 case '\t':
5077     iin.Replace('j');
5078     continue
5079 case 'X'-0x40:
5080     iin.Replace('j');
5081     continue
5082
5083 case EV_TIMEOUT:
5084     iin.Redraw();
5085     if iin.pinJmode {
5086         fprintf(stderr, "\\J\r\n")
5087         iin.inJmode = true
5088     }
5089     continue
5090 case GO_UP:
5091     if iin.lno == 1 {
5092         continue
5093     }
5094     cmd,ok := gsh.cmdStringInHistory(iin.lno-1)
5095     if ok {
5096         iin.line = cmd
5097         iin.right = ""
5098         iin.lno = iin.lno - 1
5099     }
5100     iin.Redraw();
5101     continue
5102 case GO_DOWN:
5103     cmd,ok := gsh.cmdStringInHistory(iin.lno+1)
5104     if ok {
5105         iin.line = cmd
5106         iin.right = ""
5107         iin.lno = iin.lno + 1
5108     }else{
5109         iin.line = ""
5110         iin.right = ""
5111         if iin.lno == iin.lastlno-1 {
5112             iin.lno = iin.lno + 1
5113         }
5114     }
5115     iin.Redraw();
5116     continue
5117 case GO_LEFT:
5118     if 0 < len(iin.line) {
5119         xline,tail := delTailChar(iin.line)
5120         iin.line = xline
5121         iin.right = tail + iin.right
5122     }
5123     iin.Redraw();
5124     continue;

```

```

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

```

```

5250 //== end ===== getline
5251 //
5252 // $USERHOME/.gsh/
5253 // gsh-rc.txt, or gsh-configure.txt
5254 // gsh-history.txt
5255 // gsh-aliases.txt // should be conditional?
5256 //
5257 func (gshCtx *GshContext)gshSetupHomedir()(bool) {
5258     homedir,found := userHomeDir()
5259     if !found {
5260         fmt.Printf("--E-- You have no UserHomeDir\n")
5261         return true
5262     }
5263     gshhome := homedir + "/" + GSH_HOME
5264     _, err2 := os.Stat(gshhome)
5265     if err2 != nil {
5266         err3 := os.Mkdir(gshhome,0700)
5267         if err3 != nil {
5268             fmt.Printf("--E-- Could not Create %s (%s)\n",
5269                 gshhome,err3)
5270             return true
5271         }
5272         fmt.Printf("--I-- Created %s\n",gshhome)
5273     }
5274     gshCtx.GshHomeDir = gshhome
5275     return false
5276 }
5277 func setupGshContext()(GshContext,bool){
5278     gshPA := syscall.ProcAttr {
5279         "", // the starting directory
5280         os.Environ(), // environ[]
5281         []uintptr{os.Stdin.Fd(),os.Stdout.Fd(),os.Stderr.Fd()},
5282         nil, // OS specific
5283     }
5284     cwd, _ := os.Getwd()
5285     gshCtx := GshContext {
5286         cwd, // StartDir
5287         "", // GetLine
5288         []GChdirHistory { {cwd,time.Now(),0} }, // ChdirHistory
5289         gshPA,
5290         []GCommandHistory{}, //something for invokation?
5291         GCommandHistory{}, // CmdCurrent
5292         false,
5293         []int{},
5294         syscall.Rusage{},
5295         "", // GshHomeDir
5296         Ttyid(),
5297         false,
5298         false,
5299         []PluginInfo{},
5300         []string{},
5301         "",
5302         "v",
5303         ValueStack{},
5304         GServer{"", ""}, // LastServer
5305         "", // RSERV
5306         cwd, // RND
5307         CheckSum{},
5308     }
5309     err := gshCtx.gshSetupHomedir()
5310     return gshCtx, err
5311 }
5312 func (gsh*GshContext)gshellh(gline string)(bool){
5313     ghist := gsh.CmdCurrent
5314     ghist.WorkDir,_ = os.Getwd()
5315     ghist.WorkDirX = len(gsh.ChdirHistory)-1
5316     //fmt.Printf("--D--ChdirHistory(@@)\n",len(gsh.ChdirHistory))
5317     ghist.StartAt = time.Now()
5318     rusagev1 := Getrusagev()
5319     gsh.CmdCurrent.FoundFile = []string{}
5320     fin := gsh.tgshellh(gline)
5321     rusagev2 := Getrusagev()
5322     ghist.Rusagev = RusageSubv(rusagev2,rusagev1)
5323     ghist.EndAt = time.Now()
5324     ghist.CmdLine = gline
5325     ghist.FoundFile = gsh.CmdCurrent.FoundFile
5326 }
5327 /* record it but not show in list by default
5328 if len(gline) == 0 {
5329     continue
5330 }
5331 if gline == "hi" || gline == "history" { // don't record it
5332     continue
5333 }
5334 */
5335 gsh.CommandHistory = append(gsh.CommandHistory, ghist)
5336 return fin
5337 }
5338 // <a name="main">Main loop</a>
5339 func script(gshCtxGiven *GshContext) (_ GshContext) {
5340     gshCtxBuf,err0 := setupGshContext()
5341     if err0 {
5342         return gshCtxBuf;
5343     }
5344     gshCtx := &gshCtxBuf
5345     //fmt.Printf("--I-- GSH_HOME=%s\n",gshCtx.GshHomeDir)
5346     //resmap()
5347     /*
5348     if false {
5349         gsh_getlinev, with_exgetline :=
5350             which("PATH",[string{"which","gsh-getline","-s"}])
5351         if with_exgetline {
5352             gsh_getlinev[0] = toFullpath(gsh_getlinev[0])
5353             gshCtx.GetLine = toFullpath(gsh_getlinev[0])
5354         }else{
5355             fmt.Printf("--W-- No gsh-getline found. Using internal getline.\n");
5356         }
5357     }
5358     */
5359     ghist0 := gshCtx.CmdCurrent // something special, or gshrc script, or permanent history
5360     gshCtx.CommandHistory = append(gshCtx.CommandHistory,ghist0)
5361     prevline := ""
5362     skipping := false
5363     for hix := len(gshCtx.CommandHistory); ; {
5364         gline := gshCtx.getline(hix,skipping,prevline)
5365         if skipping {
5366             if strings.Index(gline,"fi") == 0 {
5367                 fmt.Printf("fi\n");
5368                 skipping = false;
5369             }else{

```

```

5375         //fmt.Printf("%s\n",gline);
5376     }
5377     continue
5378 }
5379 if strings.Index(gline,"if") == 0 {
5380     //fmt.Printf("--D-- if start: %s\n",gline);
5381     skipping = true;
5382     continue
5383 }
5384 if false {
5385     os.Stdout.Write([]byte("gotline:"))
5386     os.Stdout.Write([]byte(gline))
5387     os.Stdout.Write([]byte("\n"))
5388 }
5389 gline = strsubst(gshCtx,gline,true)
5390 if false {
5391     fmt.Printf("fmt.Printf %%v - %v\n",gline)
5392     fmt.Printf("fmt.Printf %%s - %s\n",gline)
5393     fmt.Printf("fmt.Printf %%x - %s\n",gline)
5394     fmt.Printf("fmt.Printf %%U - %s\n",gline)
5395     fmt.Printf("Stoutt.Write -")
5396     os.Stdout.Write([]byte(gline))
5397     fmt.Printf("\n")
5398 }
5399 /*
5400 // should be cared in substitution ?
5401 if 0 < len(gline) && gline[0] == '!' {
5402     xgline, set, err := searchHistory(gshCtx,gline)
5403     if err {
5404         continue
5405     }
5406     if set {
5407         // set the line in command line editor
5408     }
5409     gline = xgline
5410 }
5411 */
5412 fin := gshCtx.gshelllh(gline)
5413 if fin {
5414     break;
5415 }
5416 prevline = gline;
5417 hix++;
5418 }
5419 return *gshCtx
5420 }
5421 func main() {
5422     gshCtxBuf := GshContext{}
5423     gsh := *gshCtxBuf
5424     argv := os.Args
5425     if 1 < len(argv) {
5426         if isin("version",argv){
5427             gsh.showVersion(argv)
5428             return
5429         }
5430         comx := isinX("-c",argv)
5431         if 0 < comx {
5432             gshCtxBuf,err := setupGshContext()
5433             gsh := *gshCtxBuf
5434             if !err {
5435                 gsh.gshellv(argv[comx+1:])
5436             }
5437             return
5438         }
5439     }
5440     if 1 < len(argv) && isin("-s",argv) {
5441     }else{
5442         gsh.showVersion(append(argv,[string{"-l","-a"}]...))
5443     }
5444     script(nil)
5445     //gshCtx := script(nil)
5446     //gshellh(gshCtx,"time")
5447 }
5448 }
5449 //</div></details>
5450 //<details id="gsh-todo"><summary>Considerations</summary><div class="gsh-src">
5451 // - inter gsh communication, possibly running in remote hosts -- to be remote shell
5452 // - merged histories of multiple parallel gsh sessions
5453 // - alias as a function or macro
5454 // - instant alias end environ export to the permanent > ~/.gsh/gsh-alias and gsh-environ
5455 // - retrieval PATH of files by its type
5456 // - gsh as an IME with completion using history and file names as dictionaies
5457 // - gsh a scheduler in precise time of within a millisecond
5458 // - all commands have its subucomand after "---" symbol
5459 // - filename expansion by "-find" command
5460 // - history of ext code and output of each commoand
5461 // - "script" output for each command by pty-tee or telnet-tee
5462 // - $BULLTIN command in PATH to show the priority
5463 // - "?" symbol in the command (not as in arguments) shows help request
5464 // - searching command with wild card like: which ssh-*
5465 // - longformat prompt after long idle time (should dismiss by BS)
5466 // - customizing by building plugin and dynamically linking it
5467 // - generating syntactic element like "if" by macro expansion (like CPP) >> alias
5468 // - "!" symbol should be used for negation, don't wast it just for job control
5469 // - don't put too long output to tty, record it into GSH_HOME/session-id/comand-id.log
5470 // - making canonical form of command at the start adding quotation or white spaces
5471 // - name(a,b,c) ... use "(" and ")" to show both delimiter and realm
5472 // - name? or name! might be useful
5473 // - htar format - packing directory contents into a single html file using data scheme
5474 // - filepath substitution shold be done by each command, expecially in case of builtins
5475 // - @N substitution for the history of working directory, and &spec for more generic ones
5476 // - @dir prefix to do the command at there, that means like (chdir @dir; command)
5477 // - GSH_PATH for plugins
5478 // - standard command output: list of data with name, size, resouce usage, modified time
5479 // - generic sort key option -nm name, -sz size, -ru rusage, -ts start-time, -tm mod-time
5480 // -wc word-count, grep match line count, ...
5481 // - standard command execution result: a list of string, -tm, -ts, -ru, -sz, ...
5482 // - -tailf-filename like tail -f filename, repeat close and open before read
5483 // - max. size and max. duration and timeout of (generated) data transfer
5484 // - auto. numbering, aliasing, IME completion of file name (especially rm of quieer name)
5485 // - IME "?" at the top of the command line means searching history
5486 // - IME %d/0x10000/ %x/ffff/
5487 // - IME ESC to go the edit mode like in vi, and use :command as :s/x/y/g to edit history
5488 // - gsh in WebAssembly
5489 // - gsh as a HTTP server of online-manual
5490 //---END--- (^-^)/ITS more</div></details>
5491 }
5492 //<span class="gsh-golang-data">
5493 }
5494 var WorldDic = //<span id="gsh-world-dic">
5495 "data:text/dic;base64,++
5496 "Ly8gTXklJTUUVuMC4wLWJg6L6e5pu4ICgyMDIwLTA4MTlhKQpzZWthaSDkuJbnlYwKa28g44GT"+
5497 "Cm5uIOOCkwpuaSDjgasKY2hpIOOBoQp0aSDjgaEKaEG44GcnNlIOOBmwpYSDjgYsKaSDj"+
5498 "gYQK";
5499 //</span>

```



```

5750 7cBBBjAdhKMPdV0/q/irWlbf44t5cNKQKwAg7DsuJzHl6Clz8bbk+1u2u78FXyWfklQ4/qY2x\
5751 tYvjx8boynW6zwc9/Ojwz7pUtVlP0nQ2UXLo8FKODmUlvuovTDjLyxcrNWHhJqWsyKrKfPs\
5752 2Jh14LpJicQXoyPnMs5fYsKeilleoG95+WCkEj3m5emcJW5b+1yHzYELxgJmDn1/HtMcK0s\
5753 aPF7Hd34FueUz8BDWovsJzXVF/xsFepLpZ/wjQ09eiH94ZwqV562+CUHV31M5yJSHfXoRHF\
5754 wKgz9FwlrTOUwJwh5+/ocSLzQ1zG52BvtG+wpqXRYEwCaRfRdb5G5D0/PySxBHakPWO\
5755 qZ9v4Y1l0uABBAkx5weq9S806+0bnwJzEXYAUvY6Ece00175A2X00UgXm2B8K6D1w66h\
5756 2CMBjAdTcrUwWYkx7Iwy4myTH9zt3R93/8X1J0ESWetyy7qFIj1odwKAmhFE2K6D1w66h\
5757 H52hWm1aLQhRQUGYz7r6yznTL87rGq40YBjg4JBWJGayRtYex14X8/xCW+rUS9L5Ye0aD6+W\
5758 8vDw0N2ZXXw7HADPZPwPdpdsLkDokEfwEMryj47E8Ea7yxxzjXm+61FwUL46h7e00d6Q+W\
5759 Wncf9B7Vh56Z3hWk7FvmlkjhJubTFKkRbag1QcWibw1LzYk16V8667FyqVOL2XoeLrbmW\
5760 PwK79U/55Bk75FSMchwhj79Y35x77q8YspVtbgS+55hdJj6V8667FyqVOL2XoeLrbmW\
5761 YkqG552p1lOK5dJzgs+2LB1B426/g+woa6e6yUOY1JzCu0G41lqY0VQFep1wU1XUL4pPR\
5762 d3G16w1V64J35xepK1N1SuBb/34wXwK1GzGz6rflBBjbbJH71lWbGDRVdb4bieXgpPbHn\
5763 NQ31gqMHZ7ETHVuxrx45+8PpQWRNd1qVfV2gBlxEF16+rqDLV82C2nVYBids2JfBpWqFz\
5764 aW8rXybgm9qXmLmChjCnvUWfKMRc2LbzJbK8mU5cm4x/2rLdJQzN1KkYyuu01pdqccFM\
5765 gKp/ahfXooV1-JToFimZujyn8P7QHmhmXdaAUEtX6cF075UUKgyq5Oz33wV/20C7b+sch\
5766 Ltnp1tH3Yw841pGt4JWanu7Pn5xwqjx84IMabBe3Q8rflZPCJfTe05F08BnAdzSFwqYfHBU\
5767 nm1dJITHChN3eSR+42Mk5KwEtsXpMe35RJTvor3r3mm49VMogfP801D191X61dvbXmkqjvb\
5768 nFDYX09UWbXUVmH9YUzLjPzxxu0fP00/pTedhod/1XXGzawFuXp6G1Lzeme2X9lbo\
5769 0XuU19f0bLAKGqha5a5NVPhxjK7X0GluOMRm+JAFefSnaKzLRhZkLYbF5ediuWc1/wD7\
5770 fd+JL72VtDPEIjgWkZj6zEP/d5durt+ZHihXkLnhs7umT011ajkkyVScenpJ1WAlAACzAE\
5771 dgY2Sx/S+nLNO0e1XVtD/SkUr+JL579vshL75z+byNS8Q2EuQ/oa3x1/FJZS/VZ30EGEbg\
5772 ePtCYCRORCR3q6Vl0pof7XExVDAaVzeGjQCEZx56CyemxZ/7CyUwar2IINzK4NOC075/\
5773 4YWRK3XuyfGj3mxt./xDbpt8u8R17f11luoF3Tqm3U17cKxfyGwVfDvwpV9R9Aeh07FRV\
5774 hUL4693pwu1Yn+FX0C+Cy0VrWxZyH/w3n7fiiibreUtSvURmitjpkWRWmPKZmHdzFCiH\
5775 dm1f6e+e1/651MCCDD2YFEL2dFycqj38aRabQSPX1sCGUCaKrDOUysazugz6z2vTfE\
5776 LLGg1XPfjYjTchKpHr+cN+r76LoL143d451+sndv9Yr4cWeg9+SrTx6G/areLXB4WX\
5777 tgzv7Wk4n+Z8f/FfzURiA3ky5Ulm9CE8N3HgLinI5ISrNy32hsXoRnTBMbVwmiP9zT7o3\
5778 j0g8vnn35zeCGY1Gcm1w2/fvicJoJyt.ieoL10xvRGHmYnZ1/IJL6W3j5y8j+711dy057\
5779 XLjDjMm+X0FOgtrugEUTDVIpFenovWAE2KAeVrAG5T3tjBQOT+5rCIU+U1BzXPIpJumPR\
5780 4YUez9wP9x1fsv/oppuyDp9nFYih91/YNXovNSd5dGG88wms31CzfrkCQUTCZSHj+vm8q\
5781 J7VXE3Xm6wqjLSrLVB668ToExtHjJ4Cdw24+uzFvsJreT11RkFoOAltZPFzD2D12QrQ\
5782 8YV88pdsboVhRLDQ6evxrEO39y4q9DQPK5Zmjjyz021LdV7yb3zfl8qmsDm0PARTVWFC3D\
5783 N1NQGW1LjEavqOmz78D2ZVemHcdpFCU8n6FPB5zKf1FPMRHE6Fo0S0AotVm/d8Vw8km71\
5784 C58YrseFuLvsplpXb79z64erdZnyuNLK1eJdaluak7Joor315x+YA9CbQBF/ck7JKHdB\
5785 E5sg690KM89pdrJd6v3vEvYhdQuc51VM9nO/QaPP3KZ1ve8zWcmJk30kx+30RK0K6IwN\
5786 blxaFhe29JqBL80f8Gkam6n59mdGp5BmUlkpmc22R7BHSKjP0kMctCk/KAM1s0JtXeJK\
5787 vqt/OzmZbn/Z51oHT3+NpGzn2eyx7uiZ0JDM9xoyTc2BTOya+vndqW3URP1jyxmDe0e1\
5788 zq4BzYggsLmphcdLIXkcmLwXskzBgwa940stveB+f7141iK3oi05Mkod+r9/12v80P9Ec3\
5789 xp7QYU8JGTmqcatO+NeY/99v3xbh+21bh03cnol1jdFCZnzkeapsDN/vjg4XP4Cb8+W9p\
5790 9zduKz2Q3fev051yqtom030pzk9E5sHOY+FXEVL50r6x5HkDFMGaadKQ3yA09Ydfdj\
5791 pp5k1jng6grnY13dfyKI5h14o0Kj1azehB9JNtWTFBACv1uIawS2xVtAhf5B05dfpseEaP\
5792 mB1fLXOm8Xmnp/fy6aVg2f55Skno2mmPF5F3gCf3o4UGS3/wI54BwVlfbVvab720b\
5793 Xx/MwGLf9ZXPMBwX5CiafjiiHyXjhsR7BkMfG8mLT+D3CdF2qod1vnn3v3d60xw7fE\
5794 koSVf0pEpKzFegJWtLd70C6dnpH7zi0z9330LHWYJulREhZ7ptxeV69XW+3Jdasmf62\
5795 IEFsY1G5j8EajZNR0adga7TeVOR2LBSccVC8Z0u5Ue1JsbpxVqHecusjRkKYLW0VSSU1ntmW\
5796 LaycEXhPswIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIK\
5797 QAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIK\
5798 QAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIK\
5799 QAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIK\
5800 QAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIK\
5801 QAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIK\
5802 QAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIKQAIK\
5803 m38w0ncAAAAASUVR0K5CYII="";
5804
5805 GshIcon="data:image/png;base64,\
5806 iVBORw0KGgoAAAANSUHEUGaAAKwAAAB/CAYAAABymylZAAAAANSR0Iars4c6QAAAHlWELm\
5807 T0UAKGAAAAGABAeAAUAAAABAAAAPgEbaAAUAAAABAAAAGoEaMAAAAABAAAIAAIpAAQAAAAB\
5808 AAAATgAAAAAAAABIAAAAAQAAAEgAAAABAAOgAQDAAAQAAAQAAACgAgEAAAAAAQAAAYgAwAE\
5809 AAAAQAHA8AAAAC6tZAAAA1wSFLZAAALWAAACMBAJgcGAAADQRJREFUEAhtn09wFNuDD\
5810 X9/b21z+iyCKK1IK1amW1j/jB6BCKstFEFth1IqRwDstOqkEunttW2FnQY1T1atInz0\
5811 amdaY6jYjOXI7kgglarVv74b3BAQPKbVAjJ3e3r94WcJpe93scmcbj784kd/ve7723+3nF\
5812 ffv+nX8SIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIA\
5813 SIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5814 IeFu62tAhEWad8ImP2wLTXtadyBmzrT+42pzRsrD3peQvXsMtrrhgNc8f8wHfUap+zyH\
5815 ZUASIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIA\
5816 hf7VweE6PyDz+0M6JmAxwznn6REVcG54SQXwhL18XtFFcuWqH7AMRgIKQAIKQAIKQAIH\
5817 Nbgem30981Ho058sa5044Umz+EzEAE/FWHXfnjsofLD/YP80ZNUKRALVAWLqEGG4C/BG\
5818 rGK0Amb/d3ucJ1WJsIyVnsIyOKAQ8FeVYnmsYJYR5F3cmUmlt6v/fwDD1QAV457EBPzYQ/\
5819 65pV5cdz46QeHyziLlZedT1K7m51Ayw2yWmXNDa8cLcplJlWw0i01U1s3d05692yZGPT\
5820 2D8BZtDXp+28XK1cYgJg9Fve6Eh5QVCInOVeQK1LKa7gpKvWbUnKnFoods814tKywGAB\
5821 e0Lc2e8+3+r+1p162LEVYnPs6SQfapwSgmw0Kf8upDWGkzEsbawPFDreUtyYU-EFwH9e\
5822 awMF19QQt4Ccz7qYoroBVF9CrE/2gnEo/ElPa4DqHalosCU4rPjzseGLNG56z10QqVrTE\
5823 rHxDxjvvnShmYsWPTq6UEZEGJES2EWmpj4sk8SVQAt/zSeLuz5aemLHz1RVkdhpAVH0\
5824 F6R5zaZf8510gmVort1SexG/oTLB9s+r5h8u0ihusYfz191fGlP2cNSyt1IA2//w008aEK\
5825 IX87zhymPTkTB3oc1bXxa/DKX3bYpoeHh686jgCSEcXugVXALy+CVN0S08Mmi9BUOOh+oIH\
5826 zFN7pHgbb3k0oJ1FO9zr7rGvN3d1QNRtq4570TShkYjsUoK6478h1pHro614d4mU7N\
5827 4t2xw1BFI1BEG60Gw2+T9JpFNKn7Uwbrmu5WgppjG7K13801ulcApe1WCLADauxgEoc1Ys4\
5828 1B7b03kKUEt04panz0+9JpONN9NsdFQmN4oxSnokzGtn+f0CANUSNm3unJgA0vhsuc6+ \
5829 CGJzDufdWGMGrf0n8Bez1JxDXyEHa+vntqfDT10QH1kVcCeJ9j5VjebVqEtoSfV1/ERXV9\
5830 74rav8+EyC/v6wf7Xamn05Kqnr60Ylem/+GqTOA6dXNtZ8wCcmh8MObur15B1b16d6Cq\
5831 RbHvm2bHr7195JV11hSpPwyEn9XhL6Jne71K1+UwQNK2HcmG5M6VzJwLU1Y1T9tUxe+E\
5832 sBongishhWort/2FWSq6wVhWvntY4r/QkuK2WP20gixhfsnYqqyZ52bBav0/Pdh1DX8Fw\
5833 1fhpLHG6f7fbh7V9CfKUXFVwH0ZYSjzc1TK3T7jWR1Jecr0HTs1rJCXhIu09B1LxKfGKC\
5834 5wZuDRZKXN211qV08pLmxu0CaZpaVgJvdL20+SUopX3S11+3idYU2NxeCDUVAJ2K0487e\
5835 Gpge4Uap7R/74/WPD77y76+3c574a/FyENPby9V/cvZPn3oYfrytV3PCJQJfAKqAet3\
5836 7Tdwc6jEqpkBy0eaEKUDXFG7FWKKh72Wx453a+vKbsbt1a7r0hPRYm9SotQ1dVotfd2/1\
5837 Tr/7DA8W5j8W8VTLKfUMtS4CzRv411a+Y8t/zd1L0now1VcTe7fDHK3+TazsTJkL12K6\
5838 C8zVlgcGfhKsrNeXnGQ9UBVoK390MPf2TUYZa8pr05RYneJKA0/hu0tNw+cc5y9264cLN\
5839 TPY0H1r+3pL9VMM1dCp6gPLLTstasnpJrc0+BiH0y9KGNrXmH4drYneZuJfavzKb1INQKZ\
5840 eQd16tyHZXGMcWc21h9JeY9+0/wj2AjrQ+hFTPFKYutZogipvsrX070z6z0w1yJ0ePsnF3\
5841 csnYwFps13RXTkH17kyCT+G8aorSt0dzVhgMI/09rVwtAhpulzsyokf99UCZDB1211HoqT\
5842 2Ywtd1sVHhX9FdrT1h1f0gMKMF/29W2qY7k7ZDUuz1butncUdLMkyR600L450y2RS1uy8E\
5843 213vexGbegY2DF3BhGAT7f3yc0VARndI0mX/886D1mVSB1TZPT58DnaCMHxwPmHaf0Mmbfm\
5844 vhdsgJNGJ7R7X80Ju841F/WeBp4PPALZG1gtD1Zu7VBNBRp9q/ubAB6EYVYLU/ulF8EXgsVc\
5845 V9eGEnbQ/DSW0YsRrIQB34E0X/ssYPD73Wk9owbTpEzP++fJtSodyoAzc6xr/ofj5wQc\
5846 BnasW4zshv+2rDy5zqQAvdp5We1t/GQSumZYM6HgmGfPJO/y4aaU+7GfYv1+LS0KWCc+3\
5847 AjzxxwVcLD+BY307RA6IHTuc8jclEpnNz5zq24PS8Xk09H9p55d2S3TmJnq8zUePTN+OL/PC\
5848 dc2P4UFahmsfn47H6RP12VnwjzrZ5LufLwLSBs0YF72KosQJYizn2FL00aGK4U6h8+BXyQ\
5849 TkVWenYqepTqZ2fth6hgq26/jB8aPKnoBo59jZLh9L+084E59USUQHk165Vw6P3njYDW\
5850 85z1R5001+qAbRR6Tso+rzbM0xwv2x0csSSmQ1/fcFY7LPdZ21Jzr5KX+C5EdLh61xYITwL\
5851 V1/nm4/cmBCw+XmNwee48E2Nelwa0fEKYUro1IDOGPL3PawpZRGfP1nhtCOXYQ5CLgPQW\
5852 RGj4fb1+LeuY3VncC8bZf4KV1sZe2fVNVs6qjsQw++Y0t29BUzqWjgWZD110BJwZ1e18VX\
5853 KEPL9fdsBxp/2Xs6sgXKM3dfCLatf8adBN1u0UNh1Afwg6Bw93sevgj1HMULPw/b14a6npg\
5854 pksWlwr06Fymn+v3MU6MwfbD3KwyWtXWjz2Ue0u4jSj+6WUyJBTLLpsV1okE327S/NjW\
5855 Mg+21W6VtW1T14TY/bUnDxmS71u9baqa20YX5DbUX129BRpGevDrDHJ51k3m3z394vqdsYp\
5856 qdnk1kqbbVhTeH161/0vu/zgszaeFLr+tnoBcXy90Xa7q7BtQ6tbuV/oyIph8z4A27\
5857 o1Ma0Q3qY2WHWct1a17ndi3Xo2P7v2p70cmEpycew8L4Q6770Veh+ZPnED+MNFY/W2\
5858 9LRATH5EJ/yQ5gf1lW7StTREMd4gAuS03T6ARSGdmX7z+/GB47ui29U0wv9JXANj711LIS\
5859 16Y+xi8s6mYcR0xul4K1ysH6Be9110YHs791/4cxvgnh2jWb1j1XxXecYQuZ0u5WDLuq\
5860 Y6XMFg2XRCb6w7rTj5NO7UzP3v91rmdN4F5eSbK0Hotab6astQot7/betUGSubEmhrc27B1\
5861 0YqH510JvclY04FkKoz+kgw+oaJDVESgeVer9PehP+SrXWnkMNLm6VpQnU1kX1zm+PveQw\
5862 h4F8J1j9WwOrt+64Stf500WEzd2G5tCd/FZS/VXH3nagrQUL+4B2j8m8Ss/Fm19D3M3Bj\
5863 Rn3kXuzqzpsZU1ecobrcjmePWhQ1abXm1gsdu1315d3LR9ywoVm9Np1kTiE/CQY1dtWZV2\
5864 8/qpxnKkvclbdveIDDt0Usc+RxdmD1pnxm1w78tYm6HGcd2f2gZnJ+8xzSRBvQ0zrEw9\
5865 H9268hVJSOHPzRd117AAPQwz7L1spLurBj0QPxybyb/8dmn2//11/qgnago2Awqf/38+WE1\
5866 14af+5Q5EMARKA02CCP2xvJNV+LMZ78Lk3V2TLWVt2n9w4/+6zgdKxJPLq7b1TADkw1p\
5867 nHs+QSI+HiEwSRPvengV20d6nf7K0tloP1dj/1kUstatCEBIEABIEABIEABIEABIEABIEAB\
5868 IEABIEABIEABIEABIEABIEABIEABIEABIEABIEABIEABIEABIEABIEABIEABIEABIEAB\
5869 IEABIEABIEABIEABIEABIEABIEABIEABIEABIEABIEABIEABIEABIEABIEABIEABIEAB\
5870 IEABIEABIEABIEABIEABIEABIEABIEABIEABIEABIEABIEABIEABIEABIEABIEABIEAB\
5871 IEABIEABIEABIEABIEABIEABIEABIEABIEABIEABIEABIEABIEABIEABIEABIEABIEAB\
5872
5873 ITSMOREQR="data:image/png;base64,\
5874 iVBORw0KGgoAAAANSUHEUGaAAAG8AAABVQMAAADYCWjAAAAB1BMVEX///9BaeFHqDaJAAAB\

```

```

5875 Hk1EQVQ4jdXTsa2EMAwGYCMX7sICkVgjXVaCBe7CarASXdallAWgS4HwM5zEVS+mvSgS+ZBQ\
5876 8gcb4BdHyzw8szMSaUBHnm+KA44QC8LDPdn8ogT4UpPgci2ji8IGF3eLwPwAhknVyvev\
5877 UEDDkaB0XZa2nJueTDzNk1qassFCkjc4nw3E1SfWgk6jU/vAkPhg0AlSFhve8Jt0dkwDhwv\
5878 YMGSSuPyWHAr19k0tKV2sb3sdwZrUCgW88g4Rp1A9s1JPv9cTp1NRD4XFKin8XaQCIw76Lzq\
5879 ZO8dhw/4+U2Gzq1S8gbqvmkfr1N6YXK8Og1D00m1GTMvzPERA8AL9vwb0iFpSoL33fsVyrL\
5880 S9wiqDznhUI138v5n783/gBuUs2eLgic8GAAAAABJRU5ErkJggg==";
5881
5882 </script>
5883
5884 <script id="gsh-script">
5885 //document.getElementById('gsh-faviconurl').href = GshIcon
5886 //document.getElementById('gsh-faviconurl').href = GshLogo
5887 document.getElementById('gsh-faviconurl').href = ITSmoreQR
5888
5889 // id of GShell HTML elemets
5890 var E_BANNER = "gsh-banner" // banner element in HTML
5891 var E_FOOTER = "gsh-footer" // footer element in HTML
5892 var E_GINDEX = "gsh-gindex" // index of Golang code of GShell
5893 var E_GOCODE = "gsh-gocode" // Golang code of GShell
5894 var E_TODO = "gsh-todo" // TODO of GShell
5895 var E_DICT = "gsh-dict" // Dictionary of GShell
5896
5897 function bannerElem(){ return document.getElementById(E_BANNER); }
5898 function bannerStyleFunc(){ return bannerElem().style; }
5899 var bannerStyle = bannerStyleFunc()
5900 bannerStyle.backgroundImage = "url("+GshLogo+")";
5901
5902 function footerElem(){ return document.getElementById(E_FOOTER); }
5903 function footerStyle(){ return footerElem().style; }
5904 footerElem().style.backgroundImage="url("+ITSmoreQR+")";
5905 //footerStyle().backgroundImage = "url("+ITSmoreQR+")";
5906
5907 function html_fold(e){
5908     if ( e.innerHTML == "Fold" ){
5909         e.innerHTML = "Unfold"
5910         document.getElementById('gsh-menu-exit').innerHTML=""
5911         document.getElementById('GshStatement').open=false
5912         GshFeatures.open = false
5913         document.getElementById('html-src').open=false
5914         document.getElementById(E_GINDEX).open=false
5915         document.getElementById(E_GOCODE).open=false
5916         document.getElementById(E_TODO).open=false
5917         document.getElementById('references').open=false
5918     }else{
5919         e.innerHTML = "Fold"
5920         document.getElementById('GshStatement').open=true
5921         GshFeatures.open = true
5922         document.getElementById(E_GINDEX).open=true
5923         document.getElementById(E_GOCODE).open=true
5924         document.getElementById(E_TODO).open=true
5925         document.getElementById('references').open=true
5926     }
5927 }
5928 function html_pure(e){
5929     if ( e.innerHTML == "Pure" ){
5930         document.getElementById('gsh').style.display=true
5931         //document.style.display = false
5932         e.innerHTML = "Unpure"
5933     }else{
5934         document.getElementById('gsh').style.display=false
5935         //document.style.display = true
5936         e.innerHTML = "Pure"
5937     }
5938 }
5939
5940 var bannerIsStopping = false
5941 //NOTE: .com/JSREF/prop_style_backgroundposition.asp
5942 function shiftBG(){
5943     bannerIsStopping = !bannerIsStopping
5944     bannerStyle.backgroundPosition = "0 0";
5945 }
5946 // status should be inherited on Window Fork(), so use the status in DOM
5947 function html_stop(e,toggle){
5948     if( toggle ){
5949         if( e.innerHTML == "Stop" ){
5950             bannerIsStopping = true
5951             e.innerHTML = "Start"
5952         }else{
5953             bannerIsStopping = false
5954             e.innerHTML = "Stop"
5955         }
5956     }else{
5957         // update JavaScript variable from DOM status
5958         if( e.innerHTML == "Stop" ){ // shown if it's running
5959             bannerIsStopping = false
5960         }else{
5961             bannerIsStopping = true
5962         }
5963     }
5964 }
5965 html_stop(document.getElementById('GshMenuStop'),false) // onInit.
5966 //html_stop(bannerElem(),false) // onInit.
5967
5968 //https://www.w3schools.com/jsref/met_win_setinterval.asp
5969 function shiftBanner(){
5970     var now = new Date().getTime();
5971     //console.log("now="+now%10)
5972     if( !bannerIsStopping ){
5973         bannerStyle.backgroundPosition = ((now/10)%100000)+" 0";
5974     }
5975 }
5976 setInterval(shiftBanner,10); // onInit.
5977
5978 // <a href="https://developer.mozilla.org/ja/docs/Web/API/Window/open">window.open()</a>
5979 // from embedded html to standalone page
5980 var MyChildren = 0
5981 function html_fork(){
5982     MyChildren += 1
5983     WinId = document.getElementById('gsh-WinId').innerHTML + "." + MyChildren;
5984     newwin = window.open("",WinId,"");
5985     src = document.getElementById("gsh");
5986     newwin.document.write("<"+html>\n");
5987     newwin.document.write("<"+span id=\"gsh\">");
5988     newwin.document.write(src.innerHTML);
5989     newwin.document.write("<"+span>"+html>\n"); // gsh span
5990     newwin.document.getElementById('gsh-menu-exit').innerHTML = "Close";
5991     newwin.document.getElementById('gsh-WinId').innerHTML = WinId;
5992     newwin.document.close();
5993     newwin.focus();
5994 }
5995 function html_close(){
5996     window.close()
5997 }
5998 function win_jump(win){
5999     //win = window.top;

```



```

6000 win = window.opener; // https://developer.mozilla.org/ja/docs/Web/API/window.opener
6001 if( win == null ){
6002     console.log("jump to window.opener("+win+") (Error)\n")
6003 }else{
6004     console.log("jump to window.opener("+win+")\n")
6005     win.focus();
6006 }
6007 }
6008
6009 // 0.2.9 2020-0902 created checksum of HTML
6010 CRC32UNIX = 0x04c11db7 // Unix cksum
6011 function byteCRC32add(bigcrc,octstr,octlen){
6012     var crc = new Int32Array(1)
6013     crc[0] = bigcrc
6014
6015     let oi = 0
6016     for( ; oi < octlen; oi++){
6017         var oct = new Int8Array(1)
6018         oct[0] = octstr[oi]
6019         for( bi = 0; bi < 8; bi++){
6020             //console.log("--CRC32 "+crc[0]+" "+oct[0].toString(16)+" ["+oi+"."+bi+"]\n")
6021             ovf1 = crc[0] < 0 ? 1 : 0
6022             ovf2 = oct[0] < 0 ? 1 : 0
6023             ovf = ovf1 ^ ovf2
6024             oct[0] <<= 1
6025             crc[0] <<= 1
6026             if( ovf ){ crc[0] ^= CRC32UNIX }
6027         }
6028     }
6029     //console.log("--CRC32 byteAdd return crc="+crc[0]+","+oi+"/"+octlen+"\n")
6030     return crc[0];
6031 }
6032 function strCRC32add(bigcrc,stri,strlen){
6033     var crc = new Uint32Array(1)
6034     crc[0] = bigcrc
6035     var code = new Uint8Array(strlen);
6036     for( i = 0; i < strlen; i++){
6037         code[i] = stri.charCodeAtAt(i) // not charAt() !!!!
6038         //console.log("=== "+code[i].toString(16)+" <== "+stri[i+"\n")
6039     }
6040     crc[0] = byteCRC32add(crc,code,strlen)
6041     //console.log("--CRC32 strAdd return crc="+crc[0)+"\n")
6042     return crc[0]
6043 }
6044 function byteCRC32end(bigcrc,len){
6045     var crc = new Uint32Array(1)
6046     crc[0] = bigcrc
6047     var slen = new Uint8Array(4)
6048     let li = 0
6049     for( ; li < 4; ){
6050         slen[li] = len
6051         li += 1
6052         len >>= 8
6053         if( len == 0 ){
6054             break
6055         }
6056     }
6057     crc[0] = byteCRC32add(crc[0],slen,li)
6058     crc[0] ^= 0xFFFFFFFF
6059     return crc[0]
6060 }
6061 function strCRC32(stri,len){
6062     var crc = new Uint32Array(1)
6063     crc[0] = 0
6064     crc[0] = strCRC32add(0,stri,len)
6065     crc[0] = byteCRC32end(crc[0],len)
6066     //console.log("--CRC32 "+crc[0]+" "+len+"\n")
6067     return crc[0]
6068 }
6069 function html_digest(){
6070     //alert("cksum="+strCRC32("",0))
6071     //alert("cksum="+strCRC32("0",1))
6072     //return
6073
6074     version = document.getElementById('gsh-version').innerHTML
6075     sfavico = document.getElementById('gsh-faviconurl').href;
6076     sbanner = document.getElementById('gsh-banner').style.backgroundImage;
6077     spositi = document.getElementById('gsh-banner').style.backgroundPosition;
6078     sfooter = document.getElementById('gsh-footer').style.backgroundImage;
6079     document.getElementById('gsh-faviconurl').href = "";
6080     document.getElementById('gsh-banner').style.backgroundImage = "";
6081     document.getElementById('gsh-banner').style.backgroundPosition = "";
6082     document.getElementById('gsh-footer').style.backgroundImage = ""
6083
6084     //html = document.getElementById("gsh").outerHTML;
6085     html = document.getElementById("gsh").innerHTML;
6086
6087     textarea = document.createElement("textarea")
6088     textarea.innerHTML = html
6089     // <a href="https://stackoverflow.com/questions/5796718/html-entity-decode">Thanks</a>
6090     text = textarea.value
6091     //textarea.destroy()
6092     text = ""
6093     + "/<"+html>\n" // lost preamble text
6094     + "<"+span id="gsh">" // lost preamble text
6095     + text
6096     + "<"+span>"+"/html>\n" // lost trail text
6097     ;
6098
6099     tlen = text.length
6100     console.log("length="+tlen+"\n"+text)
6101     alert("cksum : " + strCRC32(text,tlen) + " " + tlen + " " + version)
6102
6103     document.getElementById('gsh-faviconurl').href = sfavico;
6104     document.getElementById('gsh-banner').style.backgroundImage = sbanner;
6105     document.getElementById('gsh-banner').style.backgroundPosition = spositi;
6106     document.getElementById('gsh-footer').style.backgroundImage = sfooter;
6107 }
6108
6109 // source code viewr
6110 function frame_close(){
6111     srcframe = document.getElementById("src-frame");
6112     srcframe.innterHTML = "";
6113     //srcframe.style.cols = 1;
6114     srcframe.style.rows = 1;
6115     srcframe.style.height = 0;
6116     srcframe.style.display = false;
6117     src = document.getElementById("src-frame-textarea");
6118     src.innerHTML = ""
6119     //src.cols = 0
6120     src.rows = 0
6121     src.display = false
6122     //alert("--closed--")
6123 }
6124 //<!-- | <span onclick="html_view();">Source</span> -->

```

```

6125 //<!-- | <span onclick="frame_close();">SourceClose</span> -->
6126 //<!--| <span>Download</span> -->
6127 function frame_open(){
6128     document.getElementById('gsh-faviconurl').href = "";
6129     oldsrc = document.getElementById("GENSRC");
6130     if( oldsrc != null ){
6131         //alert("--I--(erasing old text)")
6132         oldsrc.innerHTML = "";
6133         return
6134     }else{
6135         //alert("--I--(no old text)")
6136     }
6137     banner = document.getElementById('gsh-banner').style.backgroundImage;
6138     footer = document.getElementById('gsh-footer').style.backgroundImage;
6139     document.getElementById('gsh-banner').style.backgroundImage = "";
6140     document.getElementById('gsh-banner').style.backgroundPosition = "";
6141     document.getElementById('gsh-footer').style.backgroundImage = "";
6142
6143     src = document.getElementById("gsh");
6144     srcframe = document.getElementById("src-frame");
6145     srcframe.innerHTML = "
6146 + "<"+<cite id=\"GENSRC\">\n"
6147 + "<"+<style>\n"
6148 + "#GENSRC textarea{tab-size:4;}\n"
6149 + "#GENSRC textarea{-o-tab-size:4;}\n"
6150 + "#GENSRC textarea{-moz-tab-size:4;}\n"
6151 + "#GENSRC textarea{spellcheck:false;}\n"
6152 + "</"+<style>\n"
6153 + "<"+<textarea id="src-frame-textarea" cols=100 rows=20 class="gsh-code">"
6154 + "/<"+<html>\n" // lost preamble text
6155 + "<"+<span id="gsh">" // lost preamble text
6156 + src.innerHTML
6157 + "<"+</span><"+</html>\n" // lost trail text
6158 + "<"+<textarea>\n"
6159 + "</"+<cite><!-- GENSRC -->\n";
6160
6161 //srcframe.style.cols = 80;
6162 //srcframe.style.rows = 80;
6163
6164 document.getElementById('gsh-banner').style.backgroundImage = banner;
6165 document.getElementById('gsh-footer').style.backgroundImage = footer;
6166 }
6167 function fill_CSSView(){
6168     part = document.getElementById('GshStyleDef')
6169     view = document.getElementById('gsh-style-view')
6170     view.innerHTML = "
6171 + "<"+<textarea cols=100 rows=20 class="gsh-code">"
6172 + part.innerHTML
6173 + "<"+</textarea>"
6174 }
6175 function fill_JavaScriptView(){
6176     jspart = document.getElementById('gsh-script')
6177     view = document.getElementById('gsh-script-view')
6178     view.innerHTML = "
6179 + "<"+<textarea cols=100 rows=20 class="gsh-code">"
6180 + jspart.innerHTML
6181 + "<"+</textarea>"
6182 }
6183 function fill_DataView(){
6184     part = document.getElementById('gsh-data')
6185     view = document.getElementById('gsh-data-view')
6186     view.innerHTML = "
6187 + "<"+<textarea cols=100 rows=20 class="gsh-code">"
6188 + part.innerHTML
6189 + "<"+</textarea>"
6190 }
6191 function jumpto_StyleView(){
6192     jsview = document.getElementById('html-src')
6193     jsview.open = true
6194     jsview = document.getElementById('gsh-style-frame')
6195     jsview.open = true
6196     fill_CSSView()
6197 }
6198 function jumpto_JavaScriptView(){
6199     jsview = document.getElementById('html-src')
6200     jsview.open = true
6201     jsview = document.getElementById('gsh-script-frame')
6202     jsview.open = true
6203     fill_JavaScriptView()
6204 }
6205 function jumpto_DataView(){
6206     jsview = document.getElementById('html-src')
6207     jsview.open = true
6208     jsview = document.getElementById('gsh-data-frame')
6209     jsview.open = true
6210     fill_DataView()
6211 }
6212 function jumpto_WholeView(){
6213     jsview = document.getElementById('html-src')
6214     jsview.open = true
6215     jsview = document.getElementById('gsh-whole-view')
6216     jsview.open = true
6217     frame_open()
6218 }
6219 function html_view(){
6220     html_stop();
6221
6222     banner = document.getElementById('gsh-banner').style.backgroundImage;
6223     footer = document.getElementById('gsh-footer').style.backgroundImage;
6224     document.getElementById('gsh-banner').style.backgroundImage = "";
6225     document.getElementById('gsh-banner').style.backgroundPosition = "";
6226     document.getElementById('gsh-footer').style.backgroundImage = "";
6227
6228 //srcwin = window.open("", "CodeView2", "");
6229 srcwin = window.open("", "", "");
6230 srcwin.document.write("<span id=\"gsh\">\n");
6231
6232 src = document.getElementById("gsh");
6233 srcwin.document.write("<"+<style>\n");
6234 srcwin.document.write("<textarea{tab-size:4;}\n");
6235 srcwin.document.write("<textarea{-o-tab-size:4;}\n");
6236 srcwin.document.write("<textarea{-moz-tab-size:4;}\n");
6237 srcwin.document.write("</style>\n");
6238 srcwin.document.write("<h2>\n");
6239 srcwin.document.write("<"+<span onclick=\"window.close();><Close</span> | \n");
6240 //srcwin.document.write("<"+<span onclick=\"html_stop();><Run</span>\n");
6241 srcwin.document.write("</h2>\n");
6242 srcwin.document.write("<textarea id=\"gsh-src-src\" cols=100 rows=60>");
6243 srcwin.document.write("/<"+<html>\n");
6244 srcwin.document.write("<"+<span id="gsh">");
6245 srcwin.document.write(src.innerHTML);
6246 srcwin.document.write("<"+</span><"+</html>\n");
6247 srcwin.document.write("</"+<textarea>\n");
6248
6249 document.getElementById('gsh-banner').style.backgroundImage = banner;

```

```

6250 document.getElementById('gsh-footer').style.backgroundImage = footer
6251
6252 sty = document.getElementById("GshStyleDef");
6253 srcwin.document.write("<+>style>\n");
6254 srcwin.document.write(sty.innerHTML);
6255 srcwin.document.write("<+>/style>\n");
6256
6257 run = document.getElementById("gsh-script");
6258 srcwin.document.write("<+>script>\n");
6259 srcwin.document.write(run.innerHTML);
6260 srcwin.document.write("<+>/script>\n");
6261
6262 srcwin.document.write("<+>/span><+>/html>\n"); // gsh span
6263 srcwin.document.close();
6264 srcwin.focus();
6265 }
6266 GSH = document.getElementById("gsh")
6267
6268 //GSH.onclick = "alert('Ouch!')"
6269 //GSH.css = "(background-color:#eef;)"
6270 //GSH.style = "(background-color:#eef;)"
6271 //GSH.style.display = false;
6272 //alert('Ouch0!')
6273 //GSH.style.display = true;
6274
6275 // 2020-0904 created, tentative
6276 document.addEventListener('keydown', jgshCommand);
6277 //CurElement = GshStatement
6278 CurElement = GshMenu
6279 MemElement = GshMenu
6280
6281 function nextSib(e){
6282     n = e.nextSibling;
6283     for( i = 0; i < 100; i++ ){
6284         if( n == null ){
6285             break;
6286         }
6287         if( n.nodeName == "DETAILS" ){
6288             return n;
6289         }
6290         n = n.nextSibling;
6291     }
6292     return null;
6293 }
6294 function prevSib(e){
6295     n = e.previousSibling;
6296     for( i = 0; i < 100; i++ ){
6297         if( n == null ){
6298             break;
6299         }
6300         if( n.nodeName == "DETAILS" ){
6301             return n;
6302         }
6303         n = n.previousSibling;
6304     }
6305     return null;
6306 }
6307 function setColor(e,eName,eColor){
6308     if( e.hasChildNodes() ){
6309         s = e.childNodes;
6310         if( s != null ){
6311             for( ci = 0; ci < s.length; ci++ ){
6312                 if( s[ci].nodeName == eName ){
6313                     s[ci].style.color = eColor;
6314                     //s[ci].style.backgroundColor = eColor;
6315                     break;
6316                 }
6317             }
6318         }
6319     }
6320 }
6321
6322 // https://docs.microsoft.com/en-us/previous-versions//hh781509(v=vs.85)
6323 function showCurElementPosition(){
6324     if( document.getElementById("GPos") == null ){
6325         return;
6326     }
6327     if( GPos == null ){
6328         return;
6329     }
6330     e = CurElement
6331     GPos.innerHTML =
6332     'Y='+e.getBoundingClientRect().top+" "+
6333     'X='+e.getBoundingClientRect().left+" "
6334 }
6335 function jgshCommand(event){
6336     key = event
6337     keycode = key.code
6338     GStat.style.width = window.innerWidth
6339     GStat.style.backgroundColor = "rgba(0,0,0,0.4)"
6340     if( document.getElementById("GPos") != null ){
6341         GPos.style.width = window.innerWidth
6342     }
6343     if( document.getElementById("GLog") != null ){
6344         GLog.style.width = window.innerWidth
6345         GLog.innerHTML = ""
6346     }
6347
6348     console.log("JSGsh-Key:"+keycode+"(^-^)/")
6349     if( keycode == "Digit1" ){ // fold side-bar
6350         primary.style.width = "94%"
6351         secondary.style.width = "0%"
6352         secondary.style.opacity = 0
6353         GStat.innerHTML = "[Single Column View]"
6354     }else
6355     if( keycode == "Digit2" ){ // unfold side-bar
6356         primary.style.width = "58%"
6357         secondary.style.width = "36%"
6358         secondary.style.opacity = 1
6359         GStat.innerHTML = "[Double Column View]"
6360     }else
6361     if( keycode == "KeyU" ){ // fold/unfold all
6362         html_fold(GshMenuFold);
6363         location.href = "#"+CurElement.id;
6364     }else
6365     if( keycode == "KeyO" || keycode == "ArrowRight" ){ // fold the element
6366         CurElement.open = !CurElement.open;
6367     }else
6368     if( keycode == "ArrowRight" ){ // unfold the element
6369         CurElement.open = true
6370     }else
6371     if( keycode == "ArrowLeft" ){ // unfold the element
6372         CurElement.open = false
6373     }else
6374     if( keycode == "KeyI" ){ // inspect the element

```

```

6375     e = CurElement
6376     GLog.innerHTML += "Current Element: " + e + "<br>"
6377     + "name="+e.nodeName + ", "
6378     + "id="+e.id + ", "
6379     + "children="+e.childNodes.length + ", "
6380     + "parent="+e.parentNode.id + "<br>"
6381     + "text="+e.textContent
6382     GStat.style.backgroundColor = "rgba(0,0,0,0.8)"
6383 }else
6384 if( keycode == "KeyM" ){ // memory the position
6385     MemElement = CurElement
6386 }else
6387 if( keycode == "KeyN" || keycode == "ArrowDown" ){ // next element
6388     e = nextSib(CurElement)
6389     if( e != null ){
6390         setColor(CurElement,"SUMMARY","#fff")
6391         setColor(e,"SUMMARY","#8f8") // should be complement ?
6392         CurElement = e
6393         location.href = "#"+e.id;
6394     }
6395 }else
6396 if( keycode == "KeyP" || keycode == "ArrowUp" ){ // previous element
6397     e = prevSib(CurElement)
6398     if( e != null ){
6399         setColor(CurElement,"SUMMARY","#fff")
6400         setColor(e,"SUMMARY","#8f8") // should be complement ?
6401         CurElement = e
6402         location.href = "#"+e.id;
6403     }
6404 }else
6405 if( keycode == "KeyR" ){
6406     location.reload()
6407 }else
6408 if( keycode == "KeyJ" ){
6409     GshGrid.style.top = '120px';
6410     GshGrid.innerHTML = '>_<{Down}';
6411 }else
6412 if( keycode == "KeyK" ){
6413     GshGrid.style.top = '0px';
6414     GshGrid.innerHTML = '^_^){Up}';
6415 }else
6416 if( keycode == "KeyH" ){
6417     GshGrid.style.left = '0px';
6418     GshGrid.innerHTML = '(_){Left}';
6419 }else
6420 if( keycode == "KeyL" ){
6421     GLog.innerHTML +=
6422     'screen='+screen.width+'px'+<br>'+
6423     'window='+window.innerWidth+'px'+<br>'+
6424     GshGrid.style.left = (document.documentElement.clientWidth-160).toString(10)+'px';
6425     GshGrid.innerHTML = '@_{Right}';
6426 }else
6427 if( keycode == "Keys" ){
6428     html_stop(GshMenuStop,true)
6429 }else
6430 if( keycode == "KeyF" ){
6431     html_fork()
6432 }else
6433 if( keycode == "KeyC" ){
6434     window.close()
6435 }else
6436 if( keycode == "KeyD" ){
6437     html_digest()
6438 }
6439 }
6440 showCurElementPosition();
6441 if( document.getElementById("GPos") != null ){
6442     GPos.innerHTML += "["+key.code+"] "
6443 }
6444 }
6445 //GStat.style.width = screen.width // screen.width is physical screen width ... (^-;
6446 GPos.style.width = window.innerWidth
6447 GStat.style.width = window.innerWidth
6448 GLog.style.width = window.innerWidth
6449 showCurElementPosition();
6450 GLog.innerHTML += " ("
6451 +'screen='+screen.width+'px'+', '+
6452 +'window='+window.innerWidth+'px'+')'
6453 </script>
6454
6455
6456 <!-- ##### WebCrypto ##### -->
6457 <details id="WebCrypto"><summary>WebCrypto</summary>
6458 Reference: <a href="https://mdn.github.io/dom-examples/web-crypto/encrypt-decrypt/index.html">
6459     https://mdn.github.io/dom-examples/web-crypto/encrypt-decrypt/index.html</a>
6460 <style id="web-crypto-demo-style.css">
6461 #WebCrypto { color:#080; font-size:9pt; }
6462 #rsa-oaep-message{ width:100% !important; height:24pt; color:#000 !important;
6463     border-width:2 !important; background-color:#eee !important; }
6464 #WebCrypto input{ width:50pt; background-color:#4a4; color:#fff; border-width:0; }
6465 </style>
6466
6467 <span id="web-crypto-demo.html">
6468 <section class="encrypt-decrypt rsa-oaep">
6469 <h3 class="encrypt-decrypt-heading">Web Crypto - RSA-OAEP</h3>
6470 <section class="encrypt-decrypt-controls">
6471 <p>
6472 <b>Plain text:</b><br>
6473 <input type="text" id="rsa-oaep-message" name="message"
6474     value="Hello, GShell!" style="color:#000;background-color:#fff;font-size:12pt;">
6475 </p>
6476 <p>
6477 <input class="encrypt-button" type="button" value="Encrypt"><br>
6478 <span class="ciphertext"><b>Cipher text:</b><br>
6479 <span class="ciphertext-value"></span></span>
6480 </p>
6481 <p>
6482 <input class="decrypt-button" type="button" value="Decrypt"><br>
6483 <span class="decrypted"><b>Decrypted text:</b><br>
6484 <span class="decrypted-value"></span></span>
6485 </p>
6486 <p>
6487 <input type="button" value="ShowKey" onclick="ShowKey()"><br>
6488 <span id="PublicKey">PublicKey...</span>
6489 </p>
6490 </section>
6491 </section>
6492 </span>
6493
6494 <script id="web-crypto-rsa-oaep.js">
6495 var RSAKeyPair = null;
6496 function ShowKey(){
6497     document.getElementById("PublicKey").innerHTML = RSAKeyPair.publicKey;
6498 }
6499 (() => {

```

```

6500 //Store the calculated ciphertext here, so we can decrypt the message later.
6501 let ciphertext;
6502
6503 //Fetch the contents of the "message" textbox, and encode it
6504 //in a form we can use for the encrypt operation.
6505 function getMessageEncoding() {
6506   const messageBox = document.querySelector("#rsa-oaep-message");
6507   let message = messageBox.value;
6508   let enc = new TextEncoder();
6509   return enc.encode(message);
6510 }
6511
6512 //Get the encoded message, encrypt it and display a representation
6513 //of the ciphertext in the "Ciphertext" element.
6514 async function encryptMessage(key) {
6515   let encoded = getMessageEncoding();
6516   ciphertext = await window.crypto.subtle.encrypt(
6517     {
6518       name: "RSA-OAEP"
6519     },
6520     key,
6521     encoded
6522   );
6523
6524   //let xbuffer = new Uint8Array(ciphertext, 0, 5);
6525   let xbuffer = new Uint8Array(ciphertext, 0, ciphertext.byteLength);
6526   let b = new Uint8Array(ciphertext,0,ciphertext.byteLength);
6527   //document.write("{\"b.length\":\""+b.length+"\"}")
6528   //let b64 = btoa(b);
6529   let b64 = btoa(new Uint8Array(ciphertext,0,ciphertext.byteLength));
6530   const ciphertextValue = document.querySelector(".rsa-oaep .ciphertext-value");
6531   ciphertextValue.classList.add('fade-in');
6532   ciphertextValue.addEventListener('animationend', () => {
6533     ciphertextValue.classList.remove('fade-in');
6534   });
6535   ciphertextValue.textContent =
6536     ciphertext.byteLength
6537     + " bytes "
6538     + xbuffer
6539     //+ " "
6540     //+ b + " (" + b.length + ")"
6541     //+ b64 + " (" + b64.length + ")"
6542     ;
6543   }
6544
6545 //Fetch the ciphertext and decrypt it.
6546 //Write the decrypted message into the "Decrypted" box.
6547 async function decryptMessage(key) {
6548   let decrypted = await window.crypto.subtle.decrypt(
6549     {
6550       name: "RSA-OAEP"
6551     },
6552     key,
6553     ciphertext
6554   );
6555
6556   let dec = new TextDecoder();
6557   const decryptedValue = document.querySelector(".rsa-oaep .decrypted-value");
6558   decryptedValue.classList.add('fade-in');
6559   decryptedValue.addEventListener('animationend', () => {
6560     decryptedValue.classList.remove('fade-in');
6561   });
6562   decryptedValue.textContent = dec.decode(decrypted);
6563 }
6564
6565 //Generate an encryption key pair, then set up event listeners
6566 //on the "Encrypt" and "Decrypt" buttons.
6567 window.crypto.subtle.generateKey(
6568   {
6569     name: "RSA-OAEP",
6570     // Consider using a 4096-bit key for systems that require long-term security
6571     modulusLength: 2048,
6572     publicExponent: new Uint8Array([1, 0, 1]),
6573     hash: "SHA-256",
6574   },
6575   true,
6576   ["encrypt", "decrypt"]
6577 ),then((keyPair) => {
6578   RSAKeyPair = keyPair;
6579   const encryptButton = document.querySelector(".rsa-oaep .encrypt-button");
6580   //document.getElementById('PublicKey').innerHTML = crypto.subtle.exportKey(pkcs8,keyPair.publicKey)
6581   encryptButton.addEventListener("click", () => {
6582     encryptMessage(keyPair.publicKey);
6583   });
6584
6585   const decryptButton = document.querySelector(".rsa-oaep .decrypt-button");
6586   decryptButton.addEventListener("click", () => {
6587     decryptMessage(keyPair.privateKey);
6588   });
6589 });
6590
6591 }());
6592 </script>
6593 </details>
6594
6595 *///<br></span></html>
6596

```