

```

1  /*<html>
2  <span id="gsh" data-title="GShell" data-author="sato@its-more.jp">
3  <meta charset="UTF-8">
4  <meta name="viewport" content="width=device-width, initial-scale=1.0">
5  <link rel="icon" id="GshFaviconURL" href=""/>
```

```

125 "sort" // <a href="https://golang.org/pkg/sort/">sort</a>
126 "time" // <a href="https://golang.org/pkg/time/">time</a>
127 "bufio" // <a href="https://golang.org/pkg/bufio/">bufio</a>
128 "io/ioutil" // <a href="https://golang.org/pkg/io/ioutil/">ioutil</a>
129 "os" // <a href="https://golang.org/pkg/os/">os</a>
130 "syscall" // <a href="https://golang.org/pkg/syscall/">syscall</a>
131 "plugin" // <a href="https://golang.org/pkg/plugin/">plugin</a>
132 "net" // <a href="https://golang.org/pkg/net/">net</a>
133 "net/http" // <a href="https://golang.org/pkg/net/http/">http</a>
134 // "html" // <a href="https://golang.org/pkg/html/">html</a>
135 "path/filepath" // <a href="https://golang.org/pkg/path/filepath/">filepath</a>
136 "go/types" // <a href="https://golang.org/pkg/go/types/">types</a>
137 "go/token" // <a href="https://golang.org/pkg/go/token/">token</a>
138 "encoding/base64" // <a href="https://golang.org/pkg/encoding/base64/">base64</a>
139 "unicode/utf8" // <a href="https://golang.org/pkg/unicode/utf8/">utf8</a>
140 // "gshdata" // gshell's logo and source code
141 "hash/crc32" // <a href="https://golang.org/pkg/unicode/hash/crc32/">crc32</a>
142 )
143
144 // // 2020-0906 added,
145 // // <a href="https://golang.org/cmd/cgo/">CGO</a>
146 // #include "poll.h" // <poll.h> // </poll.h> to be closed as HTML tag :-p
147 // typedef struct { struct pollfd fdv[8]; } pollFdv;
148 // int pollx(pollFdv *fdv, int nfd, int timeout){
149 //     return poll(fdv->fdv,nfds,timeout);
150 // }
151 import "C"
152
153 // // 2020-0906 added,
154 func CFPollInl(fp*os.File, timeoutUs int)(ready uintptr){
155     var fdv = C.pollFdv{}
156     var nfds = 1
157     var timeout = timeoutUs/1000
158
159     fdv.fdv[0].fd = C.int(fp.Fd())
160     fdv.fdv[0].events = C.POLLIN
161     if( 0 < EventRecvFd ){
162         fdv.fdv[1].fd = C.int(EventRecvFd)
163         fdv.fdv[1].events = C.POLLIN
164         nfds += 1
165     }
166     r := C.pollx(&fdv,C.int(nfds),C.int(timeout))
167     if( r <= 0 ){
168         return 0
169     }
170     if (int(fdv.fdv[1].revents) & int(C.POLLIN)) != 0 {
171         //fprintf(stderr,"--De-- got Event\n");
172         return uintptr(EventFdOffset + fdv.fdv[1].fd)
173     }
174     if (int(fdv.fdv[0].revents) & int(C.POLLIN)) != 0 {
175         return uintptr(NormalFdOffset + fdv.fdv[0].fd)
176     }
177     return 0
178 }
179
180 const (
181     NAME = "gsh"
182     VERSION = "0.3.9"
183     DATE = "2020-09-12"
184     AUTHOR = "SatoxITS(^-^)"
185 )
186 var (
187     GSH_HOME = ".gsh" // under home directory
188     GSH_PORT = 9999
189     MaxStreamSize = int64(128*1024*1024*1024) // 128GiB is too large?
190     PROMPT = ">"
191     LINESIZE = (8*1024)
192     PATHSEP = ":" // should be ";" in Windows
193     DIRSEP = "/" // canbe \ in Windows
194 )
195
196 // -xX logging control
197 // --A-- all
198 // --I-- info.
199 // --D-- debug
200 // --T-- time and resource usage
201 // --W-- warning
202 // --E-- error
203 // --F-- fatal error
204 // --Xn- network
205
206 // <a name="struct">Structures</a>
207 type GCommandHistory struct {
208     StartAt time.Time // command line execution started at
209     EndAt time.Time // command line execution ended at
210     ResCode int // exit code of (external command)
211     CmdError error // error string
212     OutData *os.File // output of the command
213     FoundFile []string // output - result of ufind
214     Rusagev [2]syscall.Rusage // Resource consumption, CPU time or so
215     CmdId int // maybe with identified with arguments or impact
216     // redirection commands should not be the CmdId
217     WorkDir string // working directory at start
218     WorkDirX int // index in ChdirHistory
219     CmdLine string // command line
220 }
221 type GChdirHistory struct {
222     Dir string
223     MovedAt time.Time
224     CmdIndex int
225 }
226 type CmdMode struct {
227     Background bool
228 }
229 type Event struct {
230     when time.Time
231     event int
232     evarg int64
233     CmdIndex int
234 }
235 var CmdIndex int
236 var Events []Event
237 type PluginInfo struct {
238     Spec *plugin.Plugin
239     Addr plugin.Symbol
240     Name string // maybe relative
241     Path string // this is in Plugin but hidden
242 }
243 type GServer struct {
244     host string
245     port string
246 }
247
248 // <a href="https://tools.ietf.org/html/rfc3230">Digest</a>
249 const ( // SumType

```

```

250 SUM_ITEMS = 0x000001 // items count
251 SUM_SIZE = 0x000002 // data length (simply added)
252 SUM_SIZEHASH = 0x000004 // data length (hashed sequence)
253 SUM_DATEHASH = 0x000008 // date of data (hashed sequence)
254 // also envelope attributes like time stamp can be a part of digest
255 // hashed value of sizes or mod-date of files will be useful to detect changes
256
257 SUM_WORDS = 0x000010 // word count is a kind of digest
258 SUM_LINES = 0x000020 // line count is a kind of digest
259 SUM_SUM64 = 0x000040 // simple add of bytes, useful for human too
260
261 SUM_SUM32_BITS = 0x000100 // the number of true bits
262 SUM_SUM32_2BYTE = 0x000200 // 16bits words
263 SUM_SUM32_4BYTE = 0x000400 // 32bits words
264 SUM_SUM32_8BYTE = 0x000800 // 64bits words
265
266 SUM_SUM16_BSD = 0x001000 // UNIXsum -sum -bsd
267 SUM_SUM16_SYSV = 0x002000 // UNIXsum -sum -sysv
268 SUM_UNIXFILE = 0x004000
269 SUM_CRCIEEE = 0x008000
270 )
271 type CheckSum struct {
272     Files int64 // the number of files (or data)
273     Size int64 // content size
274     Words int64 // word count
275     Lines int64 // line count
276     SumType int
277     Sum64 uint64
278     Crc32Table crc32.Table
279     Crc32Val uint32
280     Sum16 int
281     Ctime time.Time
282     Atime time.Time
283     Mtime time.Time
284     Start time.Time
285     Done time.Time
286     RusgAtStart [2]syscall.Rusage
287     RusgAtEnd [2]syscall.Rusage
288 }
289 type ValueStack [][]string
290 type GshContext struct {
291     StartDir string // the current directory at the start
292     GetLine string // gsh-getline command as a input line editor
293     ChdirHistory []GchdirHistory // the 1st entry is wd at the start
294     gshPA syscall.ProcAttr
295     CommandHistory []GCommandHistory
296     CmdCurrent GCommandHistory
297     Background bool
298     BackgroundJobs []int
299     LastRusage syscall.Rusage
300     GshHomeDir string
301     TerminalId int
302     CmdTrace bool // should be [map]
303     CmdTime bool // should be [map]
304     PluginFuncs []PluginInfo
305     iValues []string
306     iDelimiter string // field separator of print out
307     iFormat string // default print format (of integer)
308     iValStack ValueStack
309     LastServer GServer
310     RSERV string // [gsh://]host[port]
311     RWD string // remote (target, there) working directory
312     lastCheckSum CheckSum
313 }
314
315 func nsleep(ns time.Duration){
316     time.Sleep(ns)
317 }
318 func usleep(ns time.Duration){
319     nsleep(ns*1000)
320 }
321 func msleep(ns time.Duration){
322     nsleep(ns*1000000)
323 }
324 func sleep(ns time.Duration){
325     nsleep(ns*1000000000)
326 }
327
328 func strBegins(str, pat string)(bool){
329     if len(pat) <= len(str){
330         yes := str[0:len(pat)] == pat
331         //fmt.Printf("--D-- strBegins(%v,%v)=%v\n",str,pat,yes)
332         return yes
333     }
334     //fmt.Printf("--D-- strBegins(%v,%v)=%v\n",str,pat,false)
335     return false
336 }
337 func isin(what string, list []string) bool {
338     for _, v := range list {
339         if v == what {
340             return true
341         }
342     }
343     return false
344 }
345 func isinX(what string,list[]string)(int){
346     for i,v := range list {
347         if v == what {
348             return i
349         }
350     }
351     return -1
352 }
353
354 func env(opts []string) {
355     env := os.Environ()
356     if isin("-s", opts){
357         sort.Slice(env, func(i,j int) bool {
358             return env[i] < env[j]
359         })
360     }
361     for _, v := range env {
362         fmt.Printf("%v\n",v)
363     }
364 }
365
366 // - rewriting should be context dependent
367 // - should postpone until the real point of evaluation
368 // - should rewrite only known notation of symbol
369 func scanInt(str string)(val int,leng int){
370     leng = -1
371     for i,ch := range str {
372         if '0' <= ch && ch <= '9' {
373             leng = i+1
374         }else{

```

```

375         break
376     }
377 }
378 if 0 < leng {
379     ival, _ := strconv.Atoi(str[0:leng])
380     return ival, leng
381 }else{
382     return 0, 0
383 }
384 }
385 func substHistory(gshCtx *GshContext, str string, i int, rstr string)(leng int, rst string){
386     if len(str[i+1:]) == 0 {
387         return 0, rstr
388     }
389     hi := 0
390     histlen := len(gshCtx.CommandHistory)
391     if str[i+1] == '!' {
392         hi = histlen - 1
393         leng = 1
394     }else{
395         hi, leng = scanInt(str[i+1:])
396         if leng == 0 {
397             return 0, rstr
398         }
399         if hi < 0 {
400             hi = histlen + hi
401         }
402     }
403     if 0 <= hi && hi < histlen {
404         var ext byte
405         if 1 < len(str[i+leng:]) {
406             ext = str[i+leng:][1]
407         }
408         //fmt.Printf("--D-- %v(%c)\n", str[i+leng:], str[i+leng])
409         if ext == 'f' {
410             leng += 1
411             xlist := []string{}
412             list := gshCtx.CommandHistory[hi].FoundFile
413             for _, v := range list {
414                 //list[i] = escapeWhiteSP(v)
415                 xlist = append(xlist, escapeWhiteSP(v))
416             }
417             //rstr += strings.Join(list, " ")
418             rstr += strings.Join(xlist, " ")
419         }else
420         if ext == 'e' || ext == 'd' {
421             // IN0 ... workdir at the start of the command
422             leng += 1
423             rstr += gshCtx.CommandHistory[hi].WorkDir
424         }else{
425             rstr += gshCtx.CommandHistory[hi].CmdLine
426         }
427     }else{
428         leng = 0
429     }
430     return leng, rstr
431 }
432 func escapeWhiteSP(str string)(string){
433     if len(str) == 0 {
434         return "\\z" // empty, to be ignored
435     }
436     rstr := ""
437     for _, ch := range str {
438         switch ch {
439             case '\\': rstr += "\\\\"
440             case ' ': rstr += "\\s"
441             case 't': rstr += "\\t"
442             case 'r': rstr += "\\r"
443             case 'n': rstr += "\\n"
444             default: rstr += string(ch)
445         }
446     }
447     return rstr
448 }
449 func unescapeWhiteSP(str string)(string){ // strip original escapes
450     rstr := ""
451     for i := 0; i < len(str); i++ {
452         ch := str[i]
453         if ch == '\\' {
454             if i+1 < len(str) {
455                 switch str[i+1] {
456                     case 'z':
457                         continue;
458                 }
459             }
460         }
461         rstr += string(ch)
462     }
463     return rstr
464 }
465 func unescapeWhiteSPV(strv []string)([]string){ // strip original escapes
466     ustrv := []string{}
467     for _, v := range strv {
468         ustrv = append(ustrv, unescapeWhiteSP(v))
469     }
470     return ustrv
471 }
472
473 // <a name="comexpansion">str-expansion</a>
474 // - this should be a macro processor
475 func strsubst(gshCtx *GshContext, str string, histonly bool) string {
476     rbuff := []byte{}
477     if false {
478         //@@0 Unicode should be cared as a character
479         return str
480     }
481     //rstr := ""
482     inEsc := 0 // escape characer mode
483     for i := 0; i < len(str); i++ {
484         //fmt.Printf("--D--Subst %v:%v\n", i, str[i:])
485         ch := str[i]
486         if inEsc == 0 {
487             if ch == '!' {
488                 //leng, xrstr := substHistory(gshCtx, str, i, rstr)
489                 leng, rs := substHistory(gshCtx, str, i, "")
490                 if 0 < leng {
491                     //_, rs := substHistory(gshCtx, str, i, "")
492                     rbuff = append(rbuff, []byte(rs)...)
493                     i += leng
494                     //rstr = xrstr
495                     continue
496                 }
497             }
498             switch ch {
499                 case '\\': inEsc = '\\'; continue

```

```

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

```

```

625     count, err := decoder.Read(buff)
626     if count <= 0 {
627         break
628     }
629     if err != nil {
630         break
631     }
632     os.Stdout.Write(buff[0:count])
633 }
634 }
635 // lnspl [N] [-crflf][-C \\\
636 func (gshCtx *GshContext)SplitLine(argv[]string){
637     strRep := isin("-str",argv) // "..."+
638     reader := bufio.NewReaderSize(os.Stdin,64*1024)
639     ni := 0
640     toi := 0
641     for ni = 0; ; ni++ {
642         line, err := reader.ReadString('\n')
643         if len(line) <= 0 {
644             if err != nil {
645                 fmt.Fprintf(os.Stderr,"--I-- lnspl %d to %d (%v)\n",ni,toi,err)
646                 break
647             }
648         }
649         off := 0
650         ilen := len(line)
651         remlen := len(line)
652         if strRep { os.Stdout.Write([]byte("\n")) }
653         for oi := 0; 0 < remlen; oi++ {
654             olen := remlen
655             addnl := false
656             if 72 < olen {
657                 olen = 72
658                 addnl = true
659             }
660             fmt.Fprintf(os.Stderr,"--D-- write %d [%d.%d] %d %d/%d/%d\n",
661                 toi,ni,oi,off,olen,remlen,ilen)
662             toi += 1
663             os.Stdout.Write([]byte(line[0:olen]))
664             if addnl {
665                 if strRep {
666                     os.Stdout.Write([]byte("\n\n"))
667                 }else{
668                     //os.Stdout.Write([]byte("\r\n"))
669                     os.Stdout.Write([]byte("\n"))
670                     os.Stdout.Write([]byte("\n"))
671                 }
672             }
673             line = line[olen:]
674             off += olen
675             remlen -= olen
676         }
677         if strRep { os.Stdout.Write([]byte("\n")) }
678     }
679     fmt.Fprintf(os.Stderr,"--I-- lnspl %d to %d\n",ni,toi)
680 }
681 }
682 // CRC32 <a href="http://golang.jp/pkg/hash-crc32">crc32</a>
683 // 1 0000 0100 1100 0001 0001 1101 1011 0111
684 var CRC32UNIX uint32 = uint32(0x04C11DB7) // Unix cksum
685 var CRC32IEEE uint32 = uint32(0xEDB88320)
686 func byteCRC32add(crc uint32,str[]byte,len uint64)(uint32){
687     var oi uint64
688     for oi = 0; oi < len; oi++ {
689         var oct = str[oi]
690         for bi := 0; bi < 8; bi++ {
691             //fprintf(stderr,"--CRC32 %d %X (%d.%d)\n",crc,oct,oi,bi)
692             ovf1 := (crc & 0x80000000) != 0
693             ovf2 := (oct & 0x80) != 0
694             ovf := (ovf1 && !ovf2) || (!ovf1 && ovf2)
695             oct <<= 1
696             crc <<= 1
697             if ovf { crc ^= CRC32UNIX }
698         }
699     }
700     //fprintf(stderr,"--CRC32 return %d %d\n",crc,len)
701     return crc;
702 }
703 func byteCRC32end(crc uint32, len uint64)(uint32){
704     var slen = make([]byte,4)
705     var li = 0
706     for li = 0; li < 4; {
707         slen[li] = byte(len)
708         li += 1
709         len >>= 8
710         if( len == 0 ){
711             break
712         }
713     }
714     crc = byteCRC32add(crc,slen,uint64(li))
715     crc ^= 0xFFFFFFFF
716     return crc
717 }
718 func strCRC32(str string,len uint64)(crc uint32){
719     crc = byteCRC32add(0,[byte(str),len)
720     crc = byteCRC32end(crc,len)
721     //fprintf(stderr,"--CRC32 %d %d\n",crc,len)
722     return crc
723 }
724 func CRC32Finish(crc uint32, table *crc32.Table, len uint64)(uint32){
725     var slen = make([]byte,4)
726     var li = 0
727     for li = 0; li < 4; {
728         slen[li] = byte(len & 0xFF)
729         li += 1
730         len >>= 8
731         if( len == 0 ){
732             break
733         }
734     }
735     crc = crc32.Update(crc,table,slen)
736     crc ^= 0xFFFFFFFF
737     return crc
738 }
739 }
740 func (gsh*GshContext)xChecksum(path string,argv[]string, sum*Checksum)(int64){
741     if isin("-type/f",argv) && !IsRegFile(path){
742         return 0
743     }
744     if isin("-type/d",argv) && IsRegFile(path){
745         return 0
746     }
747     file, err := os.OpenFile(path,os.O_RDONLY,0)
748     if err != nil {
749         fmt.Printf("--E-- cksum %v (%v)\n",path,err)

```

```

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

```

```

875 }
876 func showFusage(dir string, fusage *fileSum){
877     bsume := float64(((fusage.Blocks-fusage.DupBlocks)/2)*1024)/1000000.0
878     //bsumdup := float64((fusage.Blocks/2)*1024)/1000000.0
879
880     fmt.Printf("%v: %v files (%vd %vs %vh) %.6f MB (%.2f MBK)\n",
881         dir,
882         fusage.Files,
883         fusage.Dirs,
884         fusage.SymLink,
885         fusage.HLinks,
886         float64(fusage.Size)/1000000.0, bsume);
887 }
888 const (
889     S_IFMT      = 0170000
890     S_IFCHR     = 0020000
891     S_IFDIR     = 0040000
892     S_IFREG     = 0100000
893     S_IFLNK     = 0120000
894     S_IFSOCK    = 0140000
895 )
896 func cumFinfo(fsum *fileSum, path string, staterr error, fstat syscall.Stat_t, argv[]string, verb bool)(*fileSum){
897     now := time.Now()
898     if time.Second <= now.Sub(fsum.nextRepo) {
899         if !fsum.nextRepo.IsZero(){
900             tstamp := now.Format(time.Stamp)
901             showFusage(tstamp, fsum)
902         }
903         fsum.nextRepo = now.Add(time.Second)
904     }
905     if staterr != nil {
906         fsum.Err += 1
907         return fsum
908     }
909     fsum.Files += 1
910     if l < fstat.Nlink {
911         // must count only once...
912         // at least ignore ones in the same directory
913         //if finfo.Mode().IsRegular() {
914         if (fstat.Mode & S_IFMT) == S_IFREG {
915             fsum.HLinks += 1
916             fsum.DupBlocks += int64(fstat.Blocks)
917             //fmt.Printf("---Dup HardLink %v %s\n", fstat.Nlink, path)
918         }
919     }
920     //fsum.Size += finfo.Size()
921     fsum.Size += fstat.Size
922     fsum.Blocks += int64(fstat.Blocks)
923     //if verb { fmt.Printf("%dBlk %s", fstat.Blocks/2, path) }
924     if isin("-ls", argv){
925         //if verb { fmt.Printf("%4d %8d ", fstat.Blksize, fstat.Blocks) }
926         // fmt.Printf("%d\t", fstat.Blocks/2)
927     }
928     //if finfo.IsDir()
929     if (fstat.Mode & S_IFMT) == S_IFDIR {
930         fsum.Dirs += 1
931     }
932     //if (finfo.Mode() & os.ModeSymlink) != 0
933     if (fstat.Mode & S_IFMT) == S_IFLNK {
934         //if verb { fmt.Printf("symlink(%v,%s)\n", fstat.Mode, finfo.Name()) }
935         //if verb { fmt.Printf("symlink(%s,%s)\n", fstat.Mode, finfo.Name()) }
936         fsum.SymLink += 1
937     }
938     return fsum
939 }
940 func (gsh*GshContext)xxFindEntv(depth int, total *fileSum, dir string, dstat syscall.Stat_t, ei int, entv []string, npatv[]string, argv[]string)(*fileSum){
941     nols := isin("-grep", argv)
942     // sort entv
943     /*
944     if isin("-t", argv){
945         sort.Slice(filev, func(i,j int) bool {
946             return 0 < filev[i].ModTime().Sub(filev[j].ModTime())
947         })
948     }
949     */
950     /*
951     if isin("-u", argv){
952         sort.Slice(filev, func(i,j int) bool {
953             return 0 < filev[i].AccTime().Sub(filev[j].AccTime())
954         })
955     }
956     if isin("-U", argv){
957         sort.Slice(filev, func(i,j int) bool {
958             return 0 < filev[i].CreateTime().Sub(filev[j].CreateTime())
959         })
960     }
961     */
962     /*
963     if isin("-S", argv){
964         sort.Slice(filev, func(i,j int) bool {
965             return filev[j].Size() < filev[i].Size()
966         })
967     }
968     */
969     for _, filename := range entv {
970         for _, npat := range npatv {
971             match := true
972             if npat == "*" {
973                 match = true
974             }else{
975                 match, _ = filepath.Match(npat, filename)
976             }
977             path := dir + DIRSEP + filename
978             if !match {
979                 continue
980             }
981             var fstat syscall.Stat_t
982             staterr := syscall.Lstat(path, &fstat)
983             if staterr != nil {
984                 if !isin("-w", argv){fmt.Printf("ufind: %v\n", staterr) }
985                 continue;
986             }
987             if isin("-du", argv) && (fstat.Mode & S_IFMT) == S_IFDIR {
988                 // should not show size of directory in "-du" mode ...
989             }else
990             if !nols && !isin("-s", argv) && (!isin("-du", argv) || isin("-a", argv)) {
991                 if isin("-du", argv) {
992                     fmt.Printf("%d\t", fstat.Blocks/2)
993                 }
994                 showFileInfo(path, argv)
995             }
996             if true { // && isin("-du", argv)
997                 total = cumFinfo(total, path, staterr, fstat, argv, false)
998             }
999             /*

```



```

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

```

```

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

```

```

1250     fullpath = dir + DIRSEP + name
1251 }
1252 fi, err := os.Stat(fullpath)
1253 if err != nil {
1254     fullpath = dir + DIRSEP + name + ".go"
1255     fi, err = os.Stat(fullpath)
1256 }
1257 if err == nil {
1258     fm := fi.Mode()
1259     if fm.IsRegular() {
1260         // R_OK=4, W_OK=2, X_OK=1, F_OK=0
1261         if syscall.Access(fullpath,5) == nil {
1262             ffullpath = fullpath
1263             ffound = true
1264             if !isin("-s", argv) {
1265                 showFileInfo(fullpath,argv)
1266             }
1267         }
1268     }
1269 }
1270 return ffullpath, ffound
1271 }
1272 func which(list string, argv []string) (fullpathv []string, itis bool){
1273     if len(argv) <= 1 {
1274         fmt.Printf("Usage: which comand [-s] [-a] [-ls]\n")
1275         return []string{"", false
1276     }
1277     path := argv[1]
1278     if strBegins(path, "/") {
1279         // should check if executable?
1280         exOK := showIfExecutable(path, "/", argv)
1281         fmt.Printf("--D-- %v exOK=%v\n", path, exOK)
1282         return []string{path}, exOK
1283     }
1284     pathenv, efound := os.LookupEnv(list)
1285     if !efound {
1286         fmt.Printf("--E-- which: no \"%s\" environment\n", list)
1287         return []string{"", false
1288     }
1289     showall := isin("-a", argv) || 0 <= strings.Index(path, "*")
1290     dirv := strings.Split(pathenv, PATHSEP)
1291     ffound := false
1292     ffullpath := path
1293     for _, dir := range dirv {
1294         if 0 <= strings.Index(path, "*") { // by wild-card
1295             list, _ := ioutil.ReadDir(dir)
1296             ffullpath, ffound = showMatchFile(list, path, dir, argv)
1297         } else {
1298             ffullpath, ffound = showIfExecutable(path, dir, argv)
1299         }
1300         //if ffound && !isin("-a", argv) {
1301         if ffound && !showall {
1302             break;
1303         }
1304     }
1305     return []string{ffullpath}, ffound
1306 }
1307 }
1308 func stripLeadingWSParg(argv []string) ([]string){
1309     for i, 0 < len(argv); {
1310         if len(argv[i]) == 0 {
1311             argv = argv[1:]
1312         } else {
1313             break
1314         }
1315     }
1316     return argv
1317 }
1318 func xEval(argv []string, nlend bool){
1319     argv = stripLeadingWSParg(argv)
1320     if len(argv) == 0 {
1321         fmt.Printf("eval [%%format] [Go-expression]\n")
1322         return
1323     }
1324     pfmt := "%v"
1325     if argv[0][0] == '%' {
1326         pfmt = argv[0]
1327         argv = argv[1:]
1328     }
1329     if len(argv) == 0 {
1330         return
1331     }
1332     gocode := strings.Join(argv, " ");
1333     //fmt.Printf("eval [%v] [%v]\n", pfmt, gocode)
1334     fset := token.NewFileSet()
1335     rval, _ := types.Eval(fset, nil, token.NoPos, gocode)
1336     fmt.Printf(pfmt, rval.Value)
1337     if nlend { fmt.Printf("\n") }
1338 }
1339 }
1340 func getval(name string) (found bool, val int) {
1341     /* should expand the name here */
1342     if name == "gsh.pid" {
1343         return true, os.Getpid()
1344     } else {
1345         if name == "gsh.ppid" {
1346             return true, os.Getppid()
1347         }
1348     }
1349     return false, 0
1350 }
1351 }
1352 func echo(argv []string, nlend bool){
1353     for ai := 1; ai < len(argv); ai++ {
1354         if 1 < ai {
1355             fmt.Printf(" ");
1356         }
1357         arg := argv[ai]
1358         found, val := getval(arg)
1359         if found {
1360             fmt.Printf("%d", val)
1361         } else {
1362             fmt.Printf("%s", arg)
1363         }
1364     }
1365     if nlend {
1366         fmt.Printf("\n");
1367     }
1368 }
1369 }
1370 func resfile() string {
1371     return "gsh.tmp"
1372 }
1373 }
1374 //var resF *File
1375 func resmap() {
1376     //_, err := os.OpenFile(resfile(), os.O_RDWR|os.O_CREATE, os.ModeAppend)

```

```

1375 // https://developpaper.com/solution-to-golang-bad-file-descriptor-problem/
1376 _, err := os.OpenFile(resfile(), os.O_RDWR|os.O_CREATE, 0600)
1377 if err != nil {
1378     fmt.Printf("refF could not open: %s\n",err)
1379 }else{
1380     fmt.Printf("refF opened\n")
1381 }
1382 }
1383 }
1384 // @@2020-0821
1385 func gshScanArg(str string,strip int)(argv []string){
1386     var si = 0
1387     var sb = 0
1388     var inBracket = 0
1389     var arg1 = make([]byte,LINESIZE)
1390     var ax = 0
1391     debug := false
1392
1393     for ; si < len(str); si++ {
1394         if str[si] != ' ' {
1395             break
1396         }
1397     }
1398     sb = si
1399     for ; si < len(str); si++ {
1400         if sb <= si {
1401             if debug {
1402                 fmt.Printf("--Da- +%d %2d-%2d %s ... %s\n",
1403                     inBracket,sb,si,arg1[0:ax],str[si:])
1404             }
1405         }
1406         ch := str[si]
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         if 0 < inBracket {
1415             if ch == '}' {
1416                 inBracket -= 1
1417                 if 0 < strip && inBracket < strip {
1418                     //fmt.Printf("stripLEV %d < %d?\n",inBracket,strip)
1419                     continue
1420                 }
1421             }
1422             arg1[ax] = ch
1423             ax += 1
1424             continue
1425         }
1426         if str[si] == ' ' {
1427             argv = append(argv,string(arg1[0:ax]))
1428             if debug {
1429                 fmt.Printf("--Da- [%v][%v-%v] %s ... %s\n",
1430                     -1+len(argv),sb,si,str[sb:si],string(str[si:]))
1431             }
1432             sb = si+1
1433             ax = 0
1434             continue
1435         }
1436         arg1[ax] = ch
1437         ax += 1
1438     }
1439     if sb < si {
1440         argv = append(argv,string(arg1[0:ax]))
1441         if debug {
1442             fmt.Printf("--Da- [%v][%v-%v] %s ... %s\n",
1443                 -1+len(argv),sb,si,string(arg1[0:ax]),string(str[si:]))
1444         }
1445     }
1446     if debug {
1447         fmt.Printf("--Da- %d [%s] => [%d]%v\n",strip,si,len(argv),argv)
1448     }
1449     return argv
1450 }
1451 }
1452 // should get stderr (into tmpfile ?) and return
1453 func (gsh*GshContext)Popen(name,mode string)(pin*os.File,pout*os.File,err bool){
1454     var pv = []int{-1,-1}
1455     syscall.Pipe(pv)
1456
1457     xarg := gshScanArg(name,1)
1458     name = strings.Join(xarg," ")
1459
1460     pin = os.NewFile(uintptr(pv[0]),"StdoutOf-"+name)
1461     pout = os.NewFile(uintptr(pv[1]),"StdinOf-"+name)
1462     fdix := 0
1463     dir := "?"
1464     if mode == "r" {
1465         dir = "<"
1466         fdix = 1 // read from the stdout of the process
1467     }else{
1468         dir = ">"
1469         fdix = 0 // write to the stdin of the process
1470     }
1471     gshPA := gsh.gshPA
1472     savfd := gshPA.Files[fdix]
1473
1474     var fd uintptr = 0
1475     if mode == "r" {
1476         fd = pout.Fd()
1477         gshPA.Files[fdix] = pout.Fd()
1478     }else{
1479         fd = pin.Fd()
1480         gshPA.Files[fdix] = pin.Fd()
1481     }
1482     // should do this by Goroutine?
1483     if false {
1484         fmt.Printf("--Ip- Opened fd[%v] %s %v\n",fd,dir,name)
1485         fmt.Printf("--RED1 [%d,%d,%d]->[%d,%d,%d]\n",
1486             os.Stdin.Fd(),os.Stdout.Fd(),os.Stderr.Fd(),
1487             pin.Fd(),pout.Fd(),pout.Fd())
1488     }
1489     savi := os.Stdin
1490     savo := os.Stdout
1491     save := os.Stderr
1492     os.Stdin = pin
1493     os.Stdout = pout
1494     os.Stderr = pout
1495     gsh.BackGround = true
1496     gsh.gshellh(name)
1497     gsh.BackGround = false
1498     os.Stdin = savi
1499     os.Stdout = savo

```

```

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

```

```

1625     if 1000000000 < elps {
1626         return fmt.Sprintf("[%5d.%02ds]", elps/1000000000, (elps%1000000000)/1000000)
1627     }else{
1628         if 1000000 < elps {
1629             return fmt.Sprintf("[%3d.%03dms]", elps/1000000, (elps%1000000)/1000)
1630         }else{
1631             return fmt.Sprintf("[%3d.%03dus]", elps/1000, (elps%1000))
1632         }
1633     }
1634 func abftime(nanos int64)(string){
1635     if 1000000000 < nanos {
1636         return fmt.Sprintf("%d.%02ds", nanos/1000000000, (nanos%1000000000)/1000000)
1637     }else{
1638         if 1000000 < nanos {
1639             return fmt.Sprintf("%d.%03dms", nanos/1000000, (nanos%1000000)/1000)
1640         }else{
1641             return fmt.Sprintf("%d.%03dus", nanos/1000, (nanos%1000))
1642         }
1643     }
1644 func absfsize(size int64)(string){
1645     fsize := float64(size)
1646     if 1024*1024*1024 < size {
1647         return fmt.Sprintf("%.2fGiB", fsize/(1024*1024*1024))
1648     }else{
1649         if 1024*1024 < size {
1650             return fmt.Sprintf("%.3fMiB", fsize/(1024*1024))
1651         }else{
1652             return fmt.Sprintf("%.3fKiB", fsize/1024)
1653         }
1654     }
1655 func absfsize(size int64)(string){
1656     fsize := float64(size)
1657     if 1024*1024*1024 < size {
1658         return fmt.Sprintf("%.2fGiB", fsize/(1024*1024*1024))
1659     }else{
1660         if 1024*1024 < size {
1661             return fmt.Sprintf("%.3fMiB", fsize/(1024*1024))
1662         }else{
1663             return fmt.Sprintf("%.3fKiB", fsize/1024)
1664         }
1665     }
1666 func absfspeed(totalB int64, ns int64)(string){
1667     MBs := (float64(totalB)/1000000) / (float64(ns)/1000000000)
1668     if 1000 <= MBs {
1669         return fmt.Sprintf("%.3fGB/s", MBs/1000)
1670     }
1671     if 1 <= MBs {
1672         return fmt.Sprintf("%.3fMB/s", MBs)
1673     }else{
1674         return fmt.Sprintf("%.3fKB/s", MBs*1000)
1675     }
1676 }
1677 func absfspeed(totalB int64, ns time.Duration)(string){
1678     MBs := (float64(totalB)/1000000) / (float64(ns)/1000000000)
1679     if 1000 <= MBs {
1680         return fmt.Sprintf("%.3fGBps", MBs/1000)
1681     }
1682     if 1 <= MBs {
1683         return fmt.Sprintf("%.3fMBps", MBs)
1684     }else{
1685         return fmt.Sprintf("%.3fKBps", MBs*1000)
1686     }
1687 }
1688 func fileRelay(what string, in*os.File, out*os.File, size int64, bsiz int)(wcount int64){
1689     Start := time.Now()
1690     buff := make([]byte, bsiz)
1691     var total int64 = 0
1692     var rem int64 = size
1693     nio := 0
1694     Prev := time.Now()
1695     var PrevSize int64 = 0
1696
1697     fmt.Printf(Elapsed(Start)+"--In- X: %s (%v/%v/%v) START\n",
1698         what, absfsize(total), size, nio)
1699
1700     for i:= 0; ; i++ {
1701         var len = bsiz
1702         if int(rem) < len {
1703             len = int(rem)
1704         }
1705         Now := time.Now()
1706         Elps := Now.Sub(Prev);
1707         if 1000000000 < Now.Sub(Prev) {
1708             fmt.Printf(Elapsed(Start)+"--In- X: %s (%v/%v/%v) %s\n",
1709                 what, absfsize(total), size, nio,
1710                 absfspeed((total-PrevSize), Elps))
1711             Prev = Now;
1712             PrevSize = total
1713         }
1714         rlen := len
1715         if in != nil {
1716             // should watch the disconnection of out
1717             rcc, err := in.Read(buff[0:rlen])
1718             if err != nil {
1719                 fmt.Printf(Elapsed(Start)+"--En- X: %s read(%v,%v)<%v\n",
1720                     what, rcc, err, in.Name())
1721                 break
1722             }
1723             rlen = rcc
1724             if string(buff[0:10]) == "(SoftEOF " {
1725                 var ecc int64 = 0
1726                 fmt.Sscanf(string(buff), "(SoftEOF %v", &ecc)
1727                 fmt.Printf(Elapsed(Start)+"--En- X: %s Recv ((SoftEOF %v))%v\n",
1728                     what, ecc, total)
1729                 if ecc == total {
1730                     break
1731                 }
1732             }
1733         }
1734
1735         wlen := rlen
1736         if out != nil {
1737             wcc, err := out.Write(buff[0:wlen])
1738             if err != nil {
1739                 fmt.Printf(Elapsed(Start)+"--En-- X: %s write(%v,%v)>%v\n",
1740                     what, wcc, err, out.Name())
1741                 break
1742             }
1743             wlen = wcc
1744         }
1745         if wlen < rlen {
1746             fmt.Printf(Elapsed(Start)+"--En- X: %s incomplete write (%v/%v)\n",
1747                 what, wlen, rlen)
1748             break;
1749         }

```

```

1750
1751     nio += 1
1752     total += int64(rlen)
1753     rem -= int64(rlen)
1754     if rem <= 0 {
1755         break
1756     }
1757 }
1758 Done := time.Now()
1759 Elps := float64(Done.Sub(Start))/1000000000 //Seconds
1760 TotalMB := float64(total)/1000000 //MB
1761 MBps := TotalMB / Elps
1762 fmt.Printf(Elapsed(Start)+"--In- X: %s (%v/%v/%v) %v %v.3fMB/s\n",
1763     what,total,size,nio,absize(total),MBps)
1764 return total
1765 }
1766 func tcpPush(clnt *os.File){
1767     // shrink socket buffer and recover
1768     usleep(100);
1769 }
1770 func (gsh*GshContext)RexecServer(argv[]string){
1771     debug := true
1772     Start0 := time.Now()
1773     Start := Start0
1774     // if local == ";" { local = "0.0.0.0:9999" }
1775     local := "0.0.0.0:9999"
1776
1777     if 0 < len(argv) {
1778         if argv[0] == "-s" {
1779             debug = false
1780             argv = argv[1:]
1781         }
1782     }
1783     if 0 < len(argv) {
1784         argv = argv[1:]
1785     }
1786     port, err := net.ResolveTCPAddr("tcp",local);
1787     if err != nil {
1788         fmt.Printf("--En- S: Address error: %s (%s)\n",local,err)
1789         return
1790     }
1791     fmt.Printf(Elapsed(Start)+"--In- S: Listening at %s...\n",local);
1792     sconn, err := net.ListenTCP("tcp", port)
1793     if err != nil {
1794         fmt.Printf(Elapsed(Start)+"--En- S: Listen error: %s (%s)\n",local,err)
1795         return
1796     }
1797
1798     reqbuf := make([]byte,LINESIZE)
1799     res := ""
1800     for {
1801         fmt.Printf(Elapsed(Start0)+"--In- S: Listening at %s...\n",local);
1802         aconn, err := sconn.AcceptTCP()
1803         Start = time.Now()
1804         if err != nil {
1805             fmt.Printf(Elapsed(Start)+"--En- S: Accept error: %s (%s)\n",local,err)
1806             return
1807         }
1808         clnt, _ := aconn.File()
1809         fd := Clnt.Fd()
1810         ar := aconn.RemoteAddr()
1811         if debug { fmt.Printf(Elapsed(Start0)+"--In- S: Accepted TCP at %s [%d] <- %v\n",
1812             local,fd,ar) }
1813         res = fmt.Sprintf("220 GShell/%s Server\r\n",VERSION)
1814         fmt.Fprintf(clnt,"%s",res)
1815         if debug { fmt.Printf(Elapsed(Start)+"--In- S: %s",res) }
1816         count, err := clnt.Read(reqbuf)
1817         if err != nil {
1818             fmt.Printf(Elapsed(Start)+"--En- C: (%v %v) %v",
1819                 count,err,string(reqbuf))
1820         }
1821         req := string(reqbuf[:count])
1822         if debug { fmt.Printf(Elapsed(Start)+"--In- C: %v",string(req)) }
1823         reqv := strings.Split(string(req),"\r")
1824         cmdv := gshScanArg(reqv[0],0)
1825         //cmdv := strings.Split(reqv[0], " ")
1826         switch cmdv[0] {
1827             case "HELO":
1828                 res = fmt.Sprintf("250 %v",req)
1829             case "GET":
1830                 // download {remotefile|-zN} [localfile]
1831                 var dsize int64 = 32*1024*1024
1832                 var bsize int = 64*1024
1833                 var fname string = ""
1834                 var in *os.File = nil
1835                 var pseudoEOF = false
1836                 if 1 < len(cmdv) {
1837                     fname = cmdv[1]
1838                     if strBegins(fname,"-z") {
1839                         fmt.Sscanf(fname[2:], "%d",&dsize)
1840                     }else
1841                     if strBegins(fname,"{") {
1842                         xin,xout,err := gsh.Popen(fname,"r")
1843                         if err {
1844                             }else{
1845                                 xout.Close()
1846                                 defer xin.Close()
1847                                 in = xin
1848                                 dsize = MaxStreamSize
1849                                 pseudoEOF = true
1850                             }
1851                         }else{
1852                             xin,err := os.Open(fname)
1853                             if err != nil {
1854                                 fmt.Printf("--En- GET (%v)\n",err)
1855                             }else{
1856                                 defer xin.Close()
1857                                 in = xin
1858                                 fi,_ := xin.Stat()
1859                                 dsize = fi.Size()
1860                             }
1861                         }
1862                     }
1863                 }
1864                 //fmt.Printf(Elapsed(Start)+"--In- GET %v:%v\n",dsize,bsize)
1865                 res = fmt.Sprintf("200 %v\r\n",dsize)
1866                 fmt.Fprintf(clnt,"%v",res)
1867                 tcpPush(clnt); // should be separated as line in receiver
1868                 fmt.Printf(Elapsed(Start)+"--In- S: %v",res)
1869                 wcount := fileRelay("SendGET",in,clnt,dsize,bsize)
1870                 if pseudoEOF {
1871                     in.Close() // pipe from the command
1872                     // show end of stream data (its size) by OOB?
1873                     SoftEOF := fmt.Sprintf("(SoftEOF %v)",wcount)
1874                     fmt.Printf(Elapsed(Start)+"--In- S: Send %v\n",SoftEOF)
1875                 }
1876             }

```

```

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

```



```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

```

2875 if len(argv) == 0 || argv[0] == "-ls" {
2876     gshCtx.whichPlugin("",argv)
2877     return nil
2878 }
2879 name := argv[0]
2880 Pin := gshCtx.whichPlugin(name,[]string{"-s"})
2881 if Pin != nil {
2882     os.Args = argv // should be recovered?
2883     Pin.Addr.(func())()
2884     return nil
2885 }
2886 sofile := toFullPath(argv[0] + ".so") // or find it by which($PATH)
2887
2888 p, err := plugin.Open(sofile)
2889 if err != nil {
2890     fmt.Printf("--E-- plugin.Open(%s)(%v)\n",sofile,err)
2891     return err
2892 }
2893 fname := "Main"
2894 f, err := p.Lookup(fname)
2895 if( err != nil ){
2896     fmt.Printf("--E-- plugin.Lookup(%s)(%v)\n",fname,err)
2897     return err
2898 }
2899 pin := PluginInfo {p,f,name,sofile}
2900 gshCtx.PluginFuncs = append(gshCtx.PluginFuncs,pin)
2901 fmt.Printf("--I-- added (%d)\n",len(gshCtx.PluginFuncs))
2902
2903 //fmt.Printf("--I-- first call(%s:%s)%v\n",sofile,fname,argv)
2904 os.Args = argv
2905 f.(func())()
2906 return err
2907 }
2908 func (gshCtx*GshContext)Args(argv[]string){
2909     for i,v := range os.Args {
2910         fmt.Printf("[%v] %v\n",i,v)
2911     }
2912 }
2913 func (gshCtx *GshContext) showVersion(argv[]string){
2914     if isin("-l",argv) {
2915         fmt.Printf("%v/%v (%v)",NAME,VERSION,DATE);
2916     }else{
2917         fmt.Printf("%v",VERSION);
2918     }
2919     if isin("-a",argv) {
2920         fmt.Printf(" %s",AUTHOR)
2921     }
2922     if !isin("-n",argv) {
2923         fmt.Printf("\n")
2924     }
2925 }
2926
2927 // <a name="scanf">Scanf</a> // string decomposer
2928 // scanf [format] [input]
2929 func scanf(sstr string)(strv[]string){
2930     strv = strings.Split(sstr," ")
2931     return strv
2932 }
2933 func scanUntil(src,end string)(rstr string,leng int){
2934     idx := strings.Index(src,end)
2935     if 0 <= idx {
2936         rstr = src[0:idx]
2937         return rstr,idx+len(end)
2938     }
2939     return src,0
2940 }
2941
2942 // -bn -- display base-name part only // can be in some %fmt, for sed rewriting
2943 func (gsh*GshContext)printVal(fmts string, vstr string, optv[]string){
2944     //vint,err := strconv.Atoi(vstr)
2945     var ival int64 = 0
2946     n := 0
2947     err := error(nil)
2948     if strBegins(vstr, "-") {
2949         vx, _ := strconv.Atoi(vstr[1:])
2950         if vx < len(gsh.iValues) {
2951             vstr = gsh.iValues[vx]
2952         }else{
2953         }
2954     }
2955     // should use Eval()
2956     if strBegins(vstr,"0x") {
2957         n,err = fmt.Sscanf(vstr[2:], "%x",&ival)
2958     }else{
2959         n,err = fmt.Sscanf(vstr,"%d",&ival)
2960     }//fmt.Printf("--D-- n=%d err=(%v) {%s}=%v\n",n,err,vstr, ival)
2961 }
2962 if n == 1 && err == nil {
2963     //fmt.Printf("--D-- formatn(%v) ival(%v)\n",fmts,ival)
2964     fmt.Printf("%"+fmts,ival)
2965 }else{
2966     if isin("-bn",optv){
2967         fmt.Printf("%"+fmts,filepath.Base(vstr))
2968     }else{
2969         fmt.Printf("%"+fmts,vstr)
2970     }
2971 }
2972 }
2973 func (gsh*GshContext)printfv(fmts,div string,argv[]string,optv[]string,list[]string){
2974     //fmt.Printf("%d",len(list))
2975     //curfmt := "v"
2976     outlen := 0
2977     curfmt := gsh.iFormat
2978
2979     if 0 < len(fmts) {
2980         for xi := 0; xi < len(fmts); xi++ {
2981             fch := fmts[xi]
2982             if fch == '%' {
2983                 if xi+1 < len(fmts) {
2984                     curfmt = string(fmts[xi+1])
2985                 }
2986                 gsh.iFormat = curfmt
2987                 xi += 1
2988                 if xi+1 < len(fmts) && fmts[xi+1] == '(' {
2989                     vals,leng := scanUntil(fmts[xi+2:],")")
2990                     //fmt.Printf("--D-- show fmt(%v) val(%v) next(%v)\n",curfmt,vals,leng)
2991                     gsh.printVal(curfmt,vals,optv)
2992                     xi += 2+leng-1
2993                 }
2994                 outlen += 1
2995             }
2996             continue
2997         }
2998     }
2999     if fch == '_' {
3000         hi,leng := scanInt(fmts[xi+1:])
3001         if 0 < leng {

```



```

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

```

```

3125     gsh.iValues = v
3126     }
3127     gsh.iValStack = gsh.iValStack[0:depth-1]
3128     fmt.Printf("depth=%d %s\n", len(gsh.iValStack), gsh.iValues)
3129 }else{
3130     fmt.Printf("depth=%d\n", depth)
3131 }
3132 }
3133
3134 // <a name="interpreter">Command Interpreter</a>
3135 func (gshCtx*GshContext)gshellv(argv []string) (fin bool) {
3136     fin = false
3137
3138     if gshCtx.CmdTrace { fmt.Fprintf(os.Stderr, "--I-- gshellv(%d)\n", len(argv)) }
3139     if len(argv) <= 0 {
3140         return false
3141     }
3142     xargv := []string{}
3143     for ai := 0; ai < len(argv); ai++ {
3144         xargv = append(xargv, strsubst(gshCtx, argv[ai], false))
3145     }
3146     argv = xargv
3147     if false {
3148         for ai := 0; ai < len(argv); ai++ {
3149             fmt.Printf("[%d] %s [%d]T\n",
3150                 ai, argv[ai], len(argv[ai]), argv[ai])
3151         }
3152     }
3153     cmd := argv[0]
3154     if gshCtx.CmdTrace { fmt.Fprintf(os.Stderr, "--I-- gshellv(%d)\n", len(argv), argv) }
3155     switch { // https://tour.golang.org/flowcontrol/11
3156     case cmd == "":
3157         gshCtx.xPwd([]string{}); // empty command
3158     case cmd == "-x":
3159         gshCtx.CmdTrace = ! gshCtx.CmdTrace
3160     case cmd == "-xt":
3161         gshCtx.CmdTime = ! gshCtx.CmdTime
3162     case cmd == "-ot":
3163         gshCtx.sconnect(true, argv)
3164     case cmd == "-ou":
3165         gshCtx.sconnect(false, argv)
3166     case cmd == "-it":
3167         gshCtx.saccept(true, argv)
3168     case cmd == "-iu":
3169         gshCtx.saccept(false, argv)
3170     case cmd == "-i" || cmd == "<" || cmd == "-o" || cmd == ">" || cmd == "-a" || cmd == ">>" || cmd == "-s" || cmd == "><":
3171         gshCtx.redirect(argv)
3172     case cmd == "|":
3173         gshCtx.fromPipe(argv)
3174     case cmd == "args":
3175         gshCtx.Args(argv)
3176     case cmd == "bg" || cmd == "-bg":
3177         rfin := gshCtx.inBackground(argv[1:])
3178         return rfin
3179     case cmd == "-bn":
3180         gshCtx.Basename(argv)
3181     case cmd == "call":
3182         /_ = gshCtx.excommand(false, argv[1:])
3183     case cmd == "cd" || cmd == "chdir":
3184         gshCtx.xChdir(argv);
3185     case cmd == "-cksum":
3186         gshCtx.xFind(argv)
3187     case cmd == "-sum":
3188         gshCtx.xFind(argv)
3189     case cmd == "-sumtest":
3190         str := ""
3191         if 1 < len(argv) { str = argv[1] }
3192         crc := strCRC32(str, uint64(len(str)))
3193         fprintf(stderr, "%v %v\n", crc, len(str))
3194     case cmd == "close":
3195         gshCtx.xClose(argv)
3196     case cmd == "gcp":
3197         gshCtx.FileCopy(argv)
3198     case cmd == "dec" || cmd == "decode":
3199         gshCtx.Dec(argv)
3200     case cmd == "#define":
3201     case cmd == "dic" || cmd == "d":
3202         xDic(argv)
3203     case cmd == "dump":
3204         gshCtx.Dump(argv)
3205     case cmd == "echo" || cmd == "e":
3206         echo(argv, true)
3207     case cmd == "enc" || cmd == "encode":
3208         gshCtx.Enc(argv)
3209     case cmd == "env":
3210         env(argv)
3211     case cmd == "eval":
3212         xEval(argv[1:], true)
3213     case cmd == "ev" || cmd == "events":
3214         dumpEvents(argv)
3215     case cmd == "exec":
3216         /_ = gshCtx.excommand(true, argv[1:])
3217         // should not return here
3218     case cmd == "exit" || cmd == "quit":
3219         // write Result code EXIT to 3>
3220         return true
3221     case cmd == "fds":
3222         // dump the attributes of fds (of other process)
3223     case cmd == "-find" || cmd == "fin" || cmd == "ufind" || cmd == "uf":
3224         gshCtx.xFind(argv[1:])
3225     case cmd == "fu":
3226         gshCtx.xFind(argv[1:])
3227     case cmd == "fork":
3228         // mainly for a server
3229     case cmd == "-gen":
3230         gshCtx.gen(argv)
3231     case cmd == "-go":
3232         gshCtx.xGo(argv)
3233     case cmd == "-grep":
3234         gshCtx.xFind(argv)
3235     case cmd == "gdeg":
3236         gshCtx.Deg(argv)
3237     case cmd == "genq":
3238         gshCtx.Enq(argv)
3239     case cmd == "gpop":
3240         gshCtx.Pop(argv)
3241     case cmd == "gpush":
3242         gshCtx.Push(argv)
3243     case cmd == "history" || cmd == "hi": // hi should be alias
3244         gshCtx.xHistory(argv)
3245     case cmd == "jobs":
3246         gshCtx.xJobs(argv)
3247     case cmd == "lisp" || cmd == "nlsp":
3248         gshCtx.SplitLine(argv)
3249     case cmd == "-ls":

```

```

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

```

```

3375 file, err := os.OpenFile(gshCtx.ttyfile(), os.O_RDWR|os.O_CREATE|os.O_TRUNC, 0600)
3376 if err != nil {
3377     fmt.Printf("--F-- cannot open %s (%s)\n", gshCtx.ttyfile(), err)
3378     return file;
3379 }
3380 return file
3381 }
3382 func (gshCtx *GshContext)getline(hix int, skipping bool, prevline string) (string) {
3383     if( skipping ) {
3384         reader := bufio.NewReaderSize(os.Stdin, LINESIZE)
3385         line, _, _ := reader.ReadLine()
3386         return string(line)
3387     }else
3388     if true {
3389         return xgetline(hix, prevline, gshCtx)
3390     }
3391     /*
3392     else
3393     if( with_exgetline && gshCtx.GetLine != "" ){
3394         //var xhix int64 = int64(hix); // cast
3395         newenv := os.Environ()
3396         newenv = append(newenv, "GSH_LINENO="+strconv.FormatInt(int64(hix), 10) )
3397
3398         tty := gshCtx.ttyline()
3399         tty.WriteString(prevline)
3400         Pa := os.ProcAttr {
3401             "", // start dir
3402             newenv, //os.Environ(),
3403             []*os.File{os.Stdin, os.Stdout, os.Stderr, tty},
3404             nil,
3405         }
3406         //fmt.Printf("--I-- getline=%s // %s\n", gsh_getlinev[0], gshCtx.GetLine)
3407         proc, err := os.StartProcess(gsh_getlinev[0], []string{"getline", "getline"}, &Pa)
3408         if err != nil {
3409             fmt.Printf("--F-- getline process error (%v)\n", err)
3410             // for ; { }
3411             return "exit (getline program failed)"
3412         }
3413         //stat, err := proc.Wait()
3414         proc.Wait()
3415         buff := make([]byte, LINESIZE)
3416         count, err := tty.Read(buff)
3417         //_, err = tty.Read(buff)
3418         //fmt.Printf("--D-- getline (%d)\n", count)
3419         if err != nil {
3420             if ! (count == 0) { // && err.String() == "EOF" } {
3421                 fmt.Printf("--E-- getline error (%s)\n", err)
3422             }
3423         }else{
3424             //fmt.Printf("--I-- getline OK \"%s\"\n", buff)
3425         }
3426         tty.Close()
3427         gline := string(buff[0:count])
3428         return gline
3429     }else
3430     /*
3431     {
3432         // if isatty {
3433         fmt.Printf("!%d", hix)
3434         fmt.Print(PROMPT)
3435         // }
3436         reader := bufio.NewReaderSize(os.Stdin, LINESIZE)
3437         line, _, _ := reader.ReadLine()
3438         return string(line)
3439     }
3440 }
3441
3442 //== begin ===== getline
3443 /*
3444 * getline.c
3445 * 2020-0819 extracted from dog.c
3446 * getline.go
3447 * 2020-0822 ported to Go
3448 */
3449 /*
3450 package main // getline main
3451 import (
3452     "fmt" // <a href="https://golang.org/pkg/fmt/">fmt</a>
3453     "strings" // <a href="https://golang.org/pkg/strings/">strings</a>
3454     "os" // <a href="https://golang.org/pkg/os/">os</a>
3455     "syscall" // <a href="https://golang.org/pkg/syscall/">syscall</a>
3456     // "bytes" // <a href="https://golang.org/pkg/os/">os</a>
3457     // "os/exec" // <a href="https://golang.org/pkg/os/">os</a>
3458 )
3459 */
3460
3461 // C language compatibility functions
3462 var errno = 0
3463 var stdin *os.File = os.Stdin
3464 var stdout *os.File = os.Stdout
3465 var stderr *os.File = os.Stderr
3466 var EOF = -1
3467 var NULL = 0
3468 type FILE os.File
3469 type StrBuff []byte
3470 var NULL_FP *os.File = nil
3471 var NULLSP = 0
3472 //var LINESIZE = 1024
3473
3474 func system(cmdstr string)(int){
3475     PA := syscall.ProcAttr {
3476         "", // the starting directory
3477         os.Environ(),
3478         [uintptr(os.Stdin.Fd(), os.Stdout.Fd(), os.Stderr.Fd())],
3479         nil,
3480     }
3481     argv := strings.Split(cmdstr, " ")
3482     pid, err := syscall.ForkExec(argv[0], argv, &PA)
3483     if( err != nil ){
3484         fmt.Printf("--E-- syscall(%v) err(%v)\n", cmdstr, err)
3485     }
3486     syscall.Wait4(pid, nil, 0, nil)
3487
3488     /*
3489     argv := strings.Split(cmdstr, " ")
3490     fmt.Fprintf(os.Stderr, "--I-- system(%v)\n", argv)
3491     //cmd := exec.Command(argv[0], ...)
3492     cmd := exec.Command(argv[0], argv[1], argv[2])
3493     cmd.Stdin = strings.NewReader("output of system")
3494     var out bytes.Buffer
3495     cmd.Stdout = &out
3496     var serr bytes.Buffer
3497     cmd.Stderr = &serr
3498     err := cmd.Run()
3499     if err != nil {

```

```

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

```

```

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

```

```

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

```

```

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

```



```

4000     xfputss(dst,stderr);
4001     return 0;
4002 }
4003
4004 //----- LINEEDIT
4005 // "?" at the top of the line means searching history
4006
4007 // should be compatilbe with Telnet
4008 const (
4009     EV_MODE     = 255
4010     EV_IDLE     = 254
4011     EV_TIMEOUT  = 253
4012
4013     GO_UP       = 252 // k
4014     GO_DOWN    = 251 // j
4015     GO_RIGHT   = 250 // l
4016     GO_LEFT    = 249 // h
4017     DEL_RIGHT  = 248 // x
4018     GO_TOPL   = 'A'-0x40 // 0
4019     GO_ENDL    = 'E'-0x40 // $
4020
4021     GO_TOPW    = 239 // b
4022     GO_ENDW    = 238 // e
4023     GO_NEXTW   = 237 // w
4024
4025     GO_FORWCH  = 229 // f
4026     GO_PAIRCH  = 228 // %
4027
4028     GO_DEL     = 219 // d
4029
4030     HI_SRCH_FW = 209 // /
4031     HI_SRCH_BK = 208 // ?
4032     HI_SRCH_RFW = 207 // n
4033     HI_SRCH_RBK = 206 // N
4034 )
4035
4036 // should return number of octets ready to be read immediately
4037 //fprintf(stderr, "\n--Select(%v %v)\n",err,r.Bits[0])
4038
4039
4040 var EventRecvFd = -1 // file descriptor
4041 var EventSendFd = -1
4042 const EventFdOffset = 1000000
4043 const NormalFdOffset = 100
4044
4045 func putEvent(event int, evarg int){
4046     if true {
4047         if EventRecvFd < 0 {
4048             var pv = [int{-1,-1}]
4049             syscall.Pipe(pv)
4050             EventRecvFd = pv[0]
4051             EventSendFd = pv[1]
4052             //fmt.Printf("--De-- EventPipe created[%v,%v]\n",EventRecvFd,EventSendFd)
4053         }
4054     }else{
4055         if EventRecvFd < 0 {
4056             // the document differs from this spec
4057             // https://golang.org/src/syscall/syscall_unix.go?s=8096:8158#L1340
4058             sv,err := syscall.Socketpair(syscall.AF_UNIX,syscall.SOCK_STREAM,0)
4059             EventRecvFd = sv[0]
4060             EventSendFd = sv[1]
4061             if err != nil {
4062                 fmt.Printf("--De-- EventSock created[%v,%v](%v)\n",
4063                     EventRecvFd,EventSendFd,err)
4064             }
4065         }
4066     }
4067     var buf = []byte{ byte(event)}
4068     n,err := syscall.Write(EventSendFd,buf)
4069     if err != nil {
4070         fmt.Printf("--De-- putEvent[%v](%3v) (%v %v)\n",EventSendFd,event,n,err)
4071     }
4072 }
4073 func ungets(str string){
4074     for _,ch := range str {
4075         putEvent(int(ch),0)
4076     }
4077 }
4078 func (gsh*GshContext)xReplay(argv []string){
4079     hix := 0
4080     tempo := 1.0
4081     xtempo := 1.0
4082     repeat := 1
4083
4084     for _,a := range argv { // tempo
4085         if strBegins(a,"x") {
4086             fmt.Sscanf(a[1:], "%f",&xtempo)
4087             tempo = 1 / xtempo
4088             //fprintf(stderr, "--Dr-- tempo=[%v]%v\n",a[2:],tempo);
4089         }else
4090         if strBegins(a,"r") { // repeat
4091             fmt.Sscanf(a[1:], "%v",&repeat)
4092         }else
4093         if strBegins(a,"l") {
4094             fmt.Sscanf(a[1:], "%d",&hix)
4095         }else{
4096             fmt.Sscanf(a, "%d",&hix)
4097         }
4098     }
4099     if hix == 0 || len(argv) <= 1 {
4100         hix = len(gsh.CommandHistory)-1
4101     }
4102     fmt.Printf("--Ir-- Replay(!%v x%v r%v)\n",hix,xtempo,repeat)
4103     //dumpEvents(hix)
4104     //gsh.xScanReplay(hix,false,repeat,tempo,argv)
4105     go gsh.xScanReplay(hix,true,repeat,tempo,argv)
4106 }
4107
4108 // <a href="https://golang.org/pkg/syscall/#FdSet">syscall.Select</a>
4109 // 2020-0827 GShell-0.2.3
4110 /*
4111 func FpollIn1(fp *os.File,usec int)(uintptr){
4112     nfd := 1
4113
4114     rdv := syscall.FdSet {}
4115     fd1 := fp.Fd()
4116     bank1 := fd1/32
4117     mask1 := int32(1 << fd1)
4118     rdv.Bits[bank1] = mask1
4119
4120     fd2 := -1
4121     bank2 := -1
4122     var mask2 int32 = 0
4123
4124     if 0 <= EventRecvFd {

```

```

4125     fd2 = EventRecvFd
4126     nfd = fd2 + 1
4127     bank2 = fd2/32
4128     mask2 = int32(1 << fd2)
4129     rdv.Bits[bank2] |= mask2
4130     //fmt.Printf("--De-- EventPoll mask added [%d][%v][%v]\n",fd2,bank2,mask2)
4131 }
4132
4133 tout := syscall.NsecToTimeval(int64(usec*1000))
4134 //n,err := syscall.Select(nfd,&rdv,nil,nil,&tout) // spec. mismatch
4135 err := syscall.Select(nfd,&rdv,nil,nil,&tout)
4136 if err != nil {
4137     //fmt.Printf("--De-- select() err(%v)\n",err)
4138 }
4139 if err == nil {
4140     if 0 <= fd2 && (rdv.Bits[bank2] & mask2) != 0 {
4141         if false {
4142             fmt.Printf("--De-- got Event\n")
4143         }
4144         return uintptr(EventFdOffset + fd2)
4145     }else{
4146         if (rdv.Bits[bank1] & mask1) != 0 {
4147             return uintptr(NormalFdOffset + fd1)
4148         }else{
4149             return 1
4150         }
4151     }else{
4152         return 0
4153     }
4154 }
4155 */
4156 func fgetcTimeout1(fp *os.File,usec int)(int){
4157     READ1:
4158     //readyFd := FpollIn1(fp,usec)
4159     readyFd := CFPollIn1(fp,usec)
4160     if readyFd < 100 {
4161         return EV_TIMEOUT
4162     }
4163
4164     var buf [1]byte
4165
4166     if EventFdOffset <= readyFd {
4167         fd := int(readyFd-EventFdOffset)
4168         _,err := syscall.Read(fd,buf[0:1])
4169         if( err != nil ){
4170             return EOF;
4171         }else{
4172             if buf[0] == EV_MODE {
4173                 recvEvent(fd)
4174                 goto READ1
4175             }
4176             return int(buf[0])
4177         }
4178     }
4179     _,err := fp.Read(buf[0:1])
4180     if( err != nil ){
4181         return EOF;
4182     }else{
4183         return int(buf[0])
4184     }
4185 }
4186 }
4187
4188 func visibleChar(ch int)(string){
4189     switch {
4190     case '!' <= ch && ch <= '-':
4191         return string(ch)
4192     }
4193     switch ch {
4194     case ' ': return "\s"
4195     case '\n': return "\n"
4196     case '\r': return "\r"
4197     case '\t': return "\t"
4198     }
4199     switch ch {
4200     case 0x00: return "NUL"
4201     case 0x07: return "BEL"
4202     case 0x08: return "BS"
4203     case 0x0E: return "SO"
4204     case 0x0F: return "SI"
4205     case 0x1B: return "ESC"
4206     case 0x7F: return "DEL"
4207     }
4208     switch ch {
4209     case EV_IDLE: return fmt.Sprintf("IDLE")
4210     case EV_MODE: return fmt.Sprintf("MODE")
4211     }
4212     return fmt.Sprintf("%X",ch)
4213 }
4214 func recvEvent(fd int){
4215     var buf = make([]byte,1)
4216     _,_ = syscall.Read(fd,buf[0:1])
4217     if( buf[0] != 0 ){
4218         romkanmode = true
4219     }else{
4220         romkanmode = false
4221     }
4222 }
4223 func (gsh*GshContext)xScanReplay(hix int,replay bool,repeat int,tempo float64,argv[string]){
4224     var Start time.Time
4225     var events = []Event{}
4226     for _,e := range Events {
4227         if hix == 0 || e.CmdIndex == hix {
4228             events = append(events,e)
4229         }
4230     }
4231     elen := len(events)
4232     if 0 < elen {
4233         if events[elen-1].event == EV_IDLE {
4234             events = events[0:elen-1]
4235         }
4236     }
4237     for r := 0; r < repeat; r++ {
4238         for i,e := range events {
4239             nano := e.when.Nanosecond()
4240             micro := nano / 1000
4241             if Start.Second() == 0 {
4242                 Start = time.Now()
4243             }
4244             diff := time.Now().Sub(Start)
4245             if replay {
4246                 if e.event != EV_IDLE {
4247                     putEvent(e.event,0)
4248                     if e.event == EV_MODE { // event with arg
4249                         putEvent(int(e.evarg),0)

```

```

4250     }
4251     }
4252     }else{
4253     fmt.Printf("%7.3fms %#-3v !%-3v [%v.%06d] %3v %02X %-4v %10.3fms\n",
4254     float64(diff)/1000000.0,
4255     i,
4256     e.CmdIndex,
4257     e.when.Format(time.Stamp),micro,
4258     e.event,e.event,visibleChar(e.event),
4259     float64(e.evarg)/1000000.0)
4260     }
4261     if e.event == EV_IDLE {
4262     d := time.Duration(float64(time.Duration(e.evarg)) * tempo)
4263     //nsleep(time.Duration(e.evarg))
4264     nsleep(d)
4265     }
4266     }
4267     }
4268 }
4269 func dumpEvents(arg []string){
4270 hix := 0
4271 if l < len(arg) {
4272     fmt.Sscanf(arg[1], "%d", &hix)
4273 }
4274 for i,e := range Events {
4275     nano := e.when.Nanosecond()
4276     micro := nano / 1000
4277     //if e.event != EV_TIMEOUT {
4278     if hix == 0 || e.CmdIndex == hix {
4279         fmt.Printf("%#-3v !%-3v [%v.%06d] %3v %02X %-4v %10.3fms\n",i,
4280         e.CmdIndex,
4281         e.when.Format(time.Stamp),micro,
4282         e.event,e.event,visibleChar(e.event),float64(e.evarg)/1000000.0)
4283     }
4284     //}
4285 }
4286 }
4287 func fgetcTimeout(fp *os.File,usec int)(int){
4288     ch := fgetcTimeout1(fp,usec)
4289     if ch != EV_TIMEOUT {
4290         now := Time.Now()
4291         if 0 < len(Events) {
4292             last := Events[len(Events)-1]
4293             dura := int64(now.Sub(last.when))
4294             Events = append(Events,Event{last.when,EV_IDLE,dura,last.CmdIndex})
4295         }
4296         Events = append(Events,Event{time.Now(),ch,0,CmdIndex})
4297     }
4298     return ch
4299 }
4300 }
4301 var TtyMaxCol = 72 // to be obtained by ioctl?
4302 var EscTimeout = (100*1000)
4303 var (
4304     MODE_VicMode    bool // vi compatible command mode
4305     MODE_ShowMode  bool
4306     romkanmode     bool // shown translation mode, the mode to be retained
4307     MODE_Recursive bool // recursive translation
4308     MODE_CapsLock  bool // software CapsLock
4309     MODE_LowerLock bool // force lower-case character lock
4310     MODE_Vinsert  int // visible insert mode, should be like "I" icon in X Window
4311     MODE_ViTrace  bool // output newline before translation
4312 )
4313 type IInput struct {
4314     lno int
4315     lastlno int
4316     pch []int // input queue
4317     prompt string
4318     line string
4319     right string
4320     inJmode bool
4321     pinJmode bool
4322     waitingMeta string // waiting meta character
4323     LastCmd string
4324 }
4325 func (iin*IInput)Getc(timeoutUs int)(int){
4326     ch1 := EOF
4327     ch2 := EOF
4328     ch3 := EOF
4329     if( 0 < len(iin.pch) ){ // deQ
4330         ch1 = iin.pch[0]
4331         iin.pch = iin.pch[1:]
4332     }else{
4333         ch1 = fgetcTimeout(stdin,timeoutUs);
4334     }
4335     if( ch1 == 033 ){ // escape sequence
4336         ch2 = fgetcTimeout(stdin,EscTimeout);
4337         if( ch2 == EV_TIMEOUT ){
4338         }else{
4339             ch3 = fgetcTimeout(stdin,EscTimeout);
4340             if( ch3 == EV_TIMEOUT ){
4341                 iin.pch = append(iin.pch,ch2) // enQ
4342             }else{
4343                 switch( ch2 ){
4344                 default:
4345                     iin.pch = append(iin.pch,ch2) // enQ
4346                     iin.pch = append(iin.pch,ch3) // enQ
4347                 case '[':
4348                     switch( ch3 ){
4349                     case 'A': ch1 = GO_UP; // ^
4350                     case 'B': ch1 = GO_DOWN; // v
4351                     case 'C': ch1 = GO_RIGHT; // >
4352                     case 'D': ch1 = GO_LEFT; // <
4353                     case '3':
4354                     }
4355                     ch4 := fgetcTimeout(stdin,EscTimeout);
4356                     if( ch4 == '-' ){
4357                         //fprintf(stderr,"x[%02X %02X %02X %02X]\n",ch1,ch2,ch3,ch4);
4358                         ch1 = DEL_RIGHT
4359                     }
4360                 case '\\':
4361                     //ch4 := fgetcTimeout(stdin,EscTimeout);
4362                     //fprintf(stderr,"y[%02X %02X %02X %02X]\n",ch1,ch2,ch3,ch4);
4363                     switch( ch3 ){
4364                     case '-': ch1 = DEL_RIGHT
4365                     }
4366                 }
4367             }
4368         }
4369     }
4370     return ch1
4371 }
4372 func (inn*IInput)clearline(){
4373     var i int
4374     fprintf(stderr, "\r");

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```



```

5000     if( ch == 'm' || ch == 'M' ){
5001         mch := fgetcTimeout(stdin,1000*1000)
5002         if mch == 'r' {
5003             romkanmode = true
5004         }else{
5005             romkanmode = false
5006         }
5007         continue
5008     }else
5009     /*
5010     if( ch == 'k' || ch == 'K' ){
5011         MODE_Recursive = IMODE_Recursive
5012         iin.Translate(cmdch);
5013         continue
5014     }else
5015     if( ch == 'j' || ch == 'J' ){
5016         iin.Translate(cmdch);
5017         continue
5018     }else
5019     if( ch == 'i' || ch == 'I' ){
5020         iin.Replace(cmdch);
5021         continue
5022     }else
5023     if( ch == 'l' || ch == 'L' ){
5024         MODE_LowerLock = IMODE_LowerLock
5025         MODE_CapsLock = false
5026         if MODE_ViTrace {
5027             fprintf(stderr,"%v\r\n",string(cmdch));
5028         }
5029         iin.Redraw();
5030         continue
5031     }else
5032     if( ch == 'u' || ch == 'U' ){
5033         MODE_CapsLock = IMODE_CapsLock
5034         MODE_LowerLock = false
5035         if MODE_ViTrace {
5036             fprintf(stderr,"%v\r\n",string(cmdch));
5037         }
5038         iin.Redraw();
5039         continue
5040     }else
5041     if( ch == 'v' || ch == 'V' ){
5042         MODE_ViTrace = IMODE_ViTrace
5043         if MODE_ViTrace {
5044             fprintf(stderr,"%v\r\n",string(cmdch));
5045         }
5046         iin.Redraw();
5047         continue
5048     }else
5049     if( ch == 'c' || ch == 'C' ){
5050         if 0 < len(iin.line) {
5051             xline,tail := delTailChar(iin.line)
5052             if len([]byte(tail)) == 1 {
5053                 ch = int(tail[0])
5054                 if( 'a' <= ch && ch <= 'z' ){
5055                     ch = ch + 'A'-'a'
5056                 }else
5057                 if( 'A' <= ch && ch <= 'Z' ){
5058                     ch = ch + 'a'-'A'
5059                 }
5060                 iin.line = xline + string(ch)
5061             }
5062         }
5063         if MODE_ViTrace {
5064             fprintf(stderr,"%v\r\n",string(cmdch));
5065         }
5066         iin.Redraw();
5067         continue
5068     }else{
5069         iin.pch = append(iin.pch,ch) // push
5070         ch = '\\'
5071     }
5072 }
5073 switch( ch ){
5074 case 'P'-0x40: ch = GO_UP
5075 case 'N'-0x40: ch = GO_DOWN
5076 case 'B'-0x40: ch = GO_LEFT
5077 case 'F'-0x40: ch = GO_RIGHT
5078 }
5079 //fprintf(stderr,"B[02X]\n",ch);
5080 switch( ch ){
5081 case 0:
5082     continue;
5083
5084 case '\t':
5085     iin.Replace('j');
5086     continue
5087 case 'X'-0x40:
5088     iin.Replace('j');
5089     continue
5090
5091 case EV_TIMEOUT:
5092     iin.Redraw();
5093     if iin.pinJmode {
5094         fprintf(stderr,"\\J\r\n")
5095         iin.inJmode = true
5096     }
5097     continue
5098 case GO_UP:
5099     if iin.lno == 1 {
5100         continue
5101     }
5102     cmd,ok := gsh.cmdStringInHistory(iin.lno-1)
5103     if ok {
5104         iin.line = cmd
5105         iin.right = ""
5106         iin.lno = iin.lno - 1
5107     }
5108     iin.Redraw();
5109     continue
5110 case GO_DOWN:
5111     cmd,ok := gsh.cmdStringInHistory(iin.lno+1)
5112     if ok {
5113         iin.line = cmd
5114         iin.right = ""
5115         iin.lno = iin.lno + 1
5116     }else{
5117         iin.line = ""
5118         iin.right = ""
5119         if iin.lno == iin.lastlno-1 {
5120             iin.lno = iin.lno + 1
5121         }
5122     }
5123     iin.Redraw();
5124     continue

```

```

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

```

```

5250 }else{
5251     fprintf(stderr, "\r\n gotline: ");
5252     trans(line);
5253     //printf("%s\n", line);
5254     printf("\n");
5255 }
5256 */
5257 }
5258 //== end ===== getline
5259
5260 //
5261 // $USERHOME/.gsh/
5262 // gsh-rc.txt, or gsh-configure.txt
5263 // gsh-history.txt
5264 // gsh-aliases.txt // should be conditional?
5265 //
5266 func (gshCtx *GshContext)gshSetupHomedir()(bool) {
5267     homedir, found := userHomeDir()
5268     if !found {
5269         fmt.Printf("--E-- You have no UserHomeDir\n")
5270         return true
5271     }
5272     gshhome := homedir + "/" + GSH_HOME
5273     _, err2 := os.Stat(gshhome)
5274     if err2 != nil {
5275         err3 := os.Mkdir(gshhome, 0700)
5276         if err3 != nil {
5277             fmt.Printf("--E-- Could not Create %s (%s)\n",
5278                 gshhome, err3)
5279             return true
5280         }
5281         fmt.Printf("--I-- Created %s\n", gshhome)
5282     }
5283     gshCtx.GshHomeDir = gshhome
5284     return false
5285 }
5286 func setupGshContext()(GshContext, bool){
5287     gshPA := syscall.ProcAttr {
5288         "", // the starting directory
5289         os.Environ(), // environ[]
5290         []uintptr{os.Stdin.Fd(), os.Stdout.Fd(), os.Stderr.Fd()},
5291         nil, // OS specific
5292     }
5293     cwd, _ := os.Getwd()
5294     gshCtx := GshContext {
5295         cwd, // StartDir
5296         "", // GetLine
5297         []GchdirHistory { {cwd, time.Now(), 0} }, // ChdirHistory
5298         gshPA,
5299         []GCommandHistory {}, // something for invokation?
5300         GCommandHistory {}, // CmdCurrent
5301         false,
5302         []int {},
5303         syscall.Rusage {},
5304         "", // GshHomeDir
5305         Ttyid(),
5306         false,
5307         false,
5308         []PluginInfo {},
5309         []string {},
5310         "",
5311         "v",
5312         ValueStack {},
5313         GServer{"", ""}, // LastServer
5314         "", // RSERV
5315         cwd, // RND
5316         CheckSum {},
5317     }
5318     err := gshCtx.gshSetupHomedir()
5319     return gshCtx, err
5320 }
5321 func (gsh *GshContext)gshellh(gline string)(bool){
5322     ghist := gsh.CmdCurrent
5323     ghist.WorkDir_ = os.Getwd()
5324     ghist.WorkDirX = len(gsh.ChdirHistory) - 1
5325     //fmt.Printf("--D--ChdirHistory(%#d)\n", len(gsh.ChdirHistory))
5326     ghist.StartAt = time.Now()
5327     rusagev1 := Getrusagev()
5328     gsh.CmdCurrent.FoundFile = []string{}
5329     fin := gsh.tgshellh(gline)
5330     rusagev2 := Getrusagev()
5331     ghist.Rusagev = RusageSubv(rusagev2, rusagev1)
5332     ghist.EndAt = time.Now()
5333     ghist.CmdLine = gline
5334     ghist.FoundFile = gsh.CmdCurrent.FoundFile
5335
5336     /* record it but not show in list by default
5337     if len(gline) == 0 {
5338         continue
5339     }
5340     if gline == "hi" || gline == "history" { // don't record it
5341         continue
5342     }
5343     */
5344     gsh.CommandHistory = append(gsh.CommandHistory, ghist)
5345     return fin
5346 }
5347 // <a name="main">Main loop</a>
5348 func script(gshCtxGiven *GshContext) (_ GshContext) {
5349     gshCtxBuf, err0 := setupGshContext()
5350     if err0 {
5351         return gshCtxBuf;
5352     }
5353     gshCtx := &gshCtxBuf
5354
5355     //fmt.Printf("--I-- GSH_HOME=%s\n", gshCtx.GshHomeDir)
5356     //resmap()
5357
5358     /*
5359     if false {
5360         gsh_getlinev, with_exgetline :=
5361             which("PATH", []string{"which", "gsh-getline", "-s"})
5362         if with_exgetline {
5363             gsh_getlinev[0] = toFullpath(gsh_getlinev[0])
5364             gshCtx.GetLine = toFullpath(gsh_getlinev[0])
5365         }else{
5366             fmt.Printf("--W-- No gsh-getline found. Using internal getline.\n");
5367         }
5368     }
5369     */
5370
5371     ghist0 := gshCtx.CmdCurrent // something special, or gshrc script, or permanent history
5372     gshCtx.CommandHistory = append(gshCtx.CommandHistory, ghist0)
5373
5374     prevline := ""

```

```

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

```



```

5625 #LineNumbered table,tr,td {
5626     margin:0;
5627     padding:4px;
5628     spacing:0;
5629     border:12px;
5630 }
5631 textarea.LineNumber {
5632     font-size:12px;
5633     font-family:monospace,Courier New;
5634     color:#282;
5635     padding:4px;
5636     text-align:right;
5637 }
5638 textarea.LineNumbered {
5639     font-size:12px;
5640     font-family:monospace,Courier New;
5641     padding:4px;
5642     wrap:off;
5643 }
5644 #RawTextViewer{
5645     z-index:0;
5646     position:fixed; top:0px; left:0px;
5647     width:100%; height:50px;
5648     overflow:auto;
5649     color:#fff; background-color:rgba(128,128,256,0.4);
5650     font-size:12px;
5651     spellcheck:false;
5652 }
5653 #RawTextViewerClose{
5654     z-index:0;
5655     position:fixed; top:-100px; left:-100px;
5656     color:#fff; background-color:rgba(128,128,256,0.4);
5657     font-size:20px; font-family:Georgia;
5658     white-space:pre;
5659 }
5660 #GShellPlane{
5661     z-index:0;
5662     position:fixed; top:0px; left:0px;
5663     width:100%; height:50px;
5664     overflow:auto;
5665     color:#fff; background-color:rgba(128,128,256,0.6);
5666     font-size:12px;
5667 }
5668 #GTop{
5669     z-index:9;
5670     opacity:1.0;
5671     position:fixed; top:0px; left:0px;
5672     width:320px; height:20px;
5673     color:#fff; background-color:rgba(0,0,0,0.4);
5674     color:#fff; font-size:12px;
5675 }
5676 #GPos{
5677     z-index:12;
5678     position:fixed; top:0px; left:0px;
5679     opacity:1.0;
5680     width:640px; height:30px;
5681     color:#fff; background-color:rgba(0,0,0,0.4);
5682     color:#fff; font-size:12px;
5683 }
5684 #GMenu{
5685     z-index:2000;
5686     position:fixed; top:250px; left:0px;
5687     opacity:1.0;
5688     width:100px; height:100px;
5689     color:#fff;
5690     color:#fff; background-color:rgba(0,0,0,0.0);
5691     color:#fff; font-size:16px; font-family:Georgia;
5692     background-repeat:no-repeat;
5693 }
5694 #GStat{
5695     z-index:8;
5696     xopacity:0.0;
5697     position:fixed; top:20px; left:0px;
5698     width:640px; height:90px;
5699     color:#fff; background-color:rgba(0,0,0,0.4);
5700     font-size:20px; font-family:Georgia;
5701 }
5702 #GLog{
5703     z-index:10;
5704     position:fixed; top:50px; left:0px;
5705     opacity:1.0;
5706     width:640px; height:60px;
5707     color:#fff; background-color:rgba(0,0,64,0.2);
5708     font-size:12px;
5709 }
5710 #GshGrid {
5711     z-index:11;
5712     xopacity:0.0;
5713     position:fixed; top:0px; left:0px;
5714     width:320px; height:30px;
5715     color:#9f9; font-size:16px;
5716 }
5717 xbody {display:none;}
5718 .gsh-link{color:green;}
5719 #gsh {border-width:1;margin:0;padding:0;}
5720 #gsh {font-family:monospace,Courier New;color:#ddf;font-size:8px;}
5721 #gsh header{height:100px;}
5722 #xgsh header{height:100px;background-image:url(GShell-Logo00.png);}
5723 #GshMenu{font-size:14pt;color:#f88;}
5724 #GshFooter{height:100px;background-size:80px;background-repeat:no-repeat;}
5725 #gsh note{color:#000;font-size:10pt;}
5726 #gsh h2{color:#24a;font-family:Georgia;font-size:18pt;}
5727 #gsh h3{color:#24a;font-family:Georgia;font-size:16pt;}
5728 #gsh details{color:#888;background-color:#fff;font-family:monospace;}
5729 #gsh summary{font-size:16pt;color:#fff;background-color:#8af;height:30px;}
5730 #gsh pre{font-size:11pt;color:#223;background-color:#faffff;}
5731 #gsh a{color:#24a;}
5732 #gsh a[name]{color:#24a;font-size:16pt;}
5733 #gsh .gsh-src{white-space:pre;font-family:monospace,Courier New;font-size:11pt;}
5734 #gsh .gsh-src{background-color:#faffff;color:#223;}
5735 #gsh-src-src{spellcheck:false}
5736 #src-frame-textarea{white-space:pre;font-family:monospace,Courier New;font-size:11pt;}
5737 #src-frame-textarea{background-color:#faffff;color:#223;}
5738 .gsh-code {white-space:pre;font-family:monospace !important;}
5739 .gsh-code {color:#088;font-size:11pt; background-color:#eef;}
5740 .gsh-golang-data {display:none;}
5741 #gsh-WinId {color:#000;font-size:14pt;}
5742
5743 .gsh-document {font-size:11pt;background-color:#fff;font-family:Georgia;}
5744 .gsh-document {color:#000;background-color:#fff !important;}
5745 .gsh-document > h2{color:#000;background-color:#fff !important;}
5746 .gsh-document details{color:#000;background-color:#fff;font-family:Georgia;}
5747 .gsh-document p{max-width:550pt;color:#000;background-color:#fff;font-family:Georgia;}
5748 .gsh-document address{width:500pt;color:#000;background-color:#fff;font-family:Georgia;}
5749

```

```

5750 @media print {
5751 #gsh_pre{font-size:11pt !important;}
5752 }
5753 </style>
5754
5755 <!--
5756 // Logo image should be drawn by JavaScript from a meta-font.
5757 // CSS seems not follow line-splitted URL
5758 -->
5759 <script id="gsh-data">
5760 //GSellLogo="QR-ITS-more.jp.png"
5761 GSellLogo="data:image/png;base64,\
5762 lVBORw0KGogaAAANSUeUgAAQAAQAAAB/CAYAAADvs3f4AAAAAXNSR0IArs4c6QAAAhLWE1m\
5763 TU0AKgAAAqABEAaAAUAAABAAAPgEBAUAAABAAAARgEoAAWAAAABAAAIAAIpAAQAAAAB\
5764 AAAATgAAAAAABIAAAAAQAAEQAAABAAOgAQDAAAAAQAAACgAAEAaaaaQAAAGCgAAQ\
5765 AAAAQAAAH8AAAAAYx1hBgAAALwSF1zAAALwAACMBAArCGAAAF3RJRFEuAhtnQuUFWNBZ\
5766 x+tTukZ3iCgg0/jY60sb8WgMzAvn7uG4+IbSTR7YnQXQDQCKj2aWNLd2MS1RkeUaPnoCdu\
5767 4iuX7j1rJz50D0Gmf2VqIBEISggCoIMMA+mu+vu//ZMD9Uldau6a2uByv91Kkrq3vdx6/q\
5768 fnXvdx8tBA85IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5769 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5770 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5771 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5772 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5773 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5774 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5775 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5776 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5777 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5778 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5779 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5780 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5781 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5782 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5783 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5784 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5785 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5786 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5787 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5788 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5789 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5790 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5791 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5792 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5793 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5794 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5795 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5796 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5797 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5798 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5799 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5800 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5801 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5802 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5803 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5804 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5805 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5806 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5807 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5808 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5809 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5810 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5811 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5812 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5813 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5814 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5815 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5816 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5817 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5818 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5819 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5820 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5821 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5822 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5823 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5824 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5825 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5826 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5827 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5828 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5829 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5830 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5831 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5832 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5833 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5834 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5835 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5836 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5837 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5838 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5839 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5840 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5841 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5842 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5843 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5844 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5845 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5846 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5847 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5848 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5849 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5850 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5851 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5852 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5853 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5854 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5855 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5856 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5857 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5858 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5859 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5860 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5861 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5862 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5863 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5864 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5865 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5866 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5867 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5868 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5869 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5870 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5871 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5872 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5873 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\
5874 IAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAES\

```







```

6000 vhdSgJNGjXhr80QJ0u841F/WeBpAPPALZGLgtD1Zu7VBWBp9g/ubAB6EYVKYL/ulF8EXgsvC\
6001 V9eGEnDQ/DSrW0YsRXRiQB34E0x/ssYpD73WK9owbTpEezP++jFTSogyoAzc6XR/ofj5QrDY\
6002 BnasW4zhsV+2rDY55zQAVdp5WeL/GOsumZiW6HgmGfPJR0/y4aaU+7GfFy1+LS0KkwcC+3\
6003 AjxxwWLD+BYj07RA6LHTuc8jclEpNnz5z4F58XK09B9p55d2S3TmngUzUPTN+OL/PC\
6004 dc2P4Ufahmsfn47H6RP12Vnwjzrz5LuLwLBS0H772KooSQJYizn2FL00agK4U6b8+BXyQ\
6005 TkKvwenTgeupTgZ2fH6Ghg26/jB8aPKnoBo59jZLh9L+084E59SUQhki65Vwg6P3njyDw\
6006 85ziR5001+qabrR6F50+rzbMqxw2x0csSSmQL/fCFY7LPdZ2LJzrSK+C5dELh6ixITTW\
6007 V1/nm4/cmCW+XmNwee48EznelWaoF+SKyUro1IDOGPL3Pawp2RGPpLnIhtCOXYQ5ClgPQW\
6008 RCj4fbl+LEuV3VncC8bZf4KvIze2fVNVs6qj8sQv++Y0t29BUzqWzrjIHMULPw/b14a6npg\
6009 KEPL9fdsBxp/2K6sgXKM3dfClatfdAdBN1U0UNh1AEwg6Bw93sevejIHMULPw/b14a6npg\
6010 KgsWlw06fEYm+v3MLU6Mfwfbd3KvyWtXwj2zUcu04jSj+6WUyJBTlljpsv1okE327S/NdWx\
6011 Mg+21W6VtW1T4TY/bUnDxms71u9baga20YX5DbUX1z9BRpGEVdrDHJ51k3m3z394VgdsYp\
6012 qZnk1kQbbVhBteHI61/0vu/zgszaeLR+tnOBCXy90Ka7q7BbtQ6tbuV/oYiPhu8xhzA4R\
6013 o1MaOu3Q4pY2WHwCt1a17Ndi3bXo2P7v2p70cmEfyecw8L4Q6770E+htzPnED+mNFY/W2\
6014 9LRArH5EL/05gFllw7StTREM44Gu5+Q3T6aRSqdmx7z+/GB47ui290Uwv9JXAnJ711Is\
6015 16Y+xi8smf6YcrXul4KlySH6Be9110YHs791/4cxvbnH2jWB1j1XxexYQzU0g5WdLuq\
6016 Y6xMFG2XRebawTrTjP5NO7ZuP3v9lrmdnN4F5eSkOHOtAb6aStQot7/beUgSubPmhc27B\
6017 0YQh510JvclY04FkKoz+kgv+oaJdVEsgVer9PehP+SrXWnkMNLm6VpOnUikXlzm+PveQgF\
6018 h4F8J1j9wWOrt+64Stf500WEzd2G5tCd/FZS/VXH3nagrQUL+4B2j8m8SsS/FmI9D3McBjwo\
6019 KRN3KXzuzgppsZKZU1cbOCRjMPhQlAbXm1GsdU1315d3J1R9y0W9m9p1kTie/CQYIdtW2VZ\
6020 8/KpxqXKvcIbdveIDDt0Usc+RxsmdDpxnmLw78tYm6HZCdCt2fGzn+8zsuSRBV04zrhEw9\
6021 H9269VJSOHFzRdII7AAPQwzKI7LeplurBj0Qpxybyb/8dmn2//ll/qgnago2Awgf/38+NE\
6022 I4af5Q5EXMARKAoI2CCP2xjNV+LMZ78LkH3V27LWv2n9W4/+63gdkXJPLq7b1TADkw1p\
6023 nIhS+QSt+HiEW5RPUvengV20d6Nf7K0t0f1dJ/jkUsatCEBEiABEiABEiABEiABEiAB\
6024 EiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiAB\
6025 EiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiAB\
6026 EiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiAB\
6027 EiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiABEiAB\
6028
6029 ITSmoreQR="data:image/png;base64,\
6030 iVBORw0KGgoAAANSUHEUGAAAGSAAABVACMAAADYCVwJAAAAB1BMVEX///9BaePhQDaJAAAB\
6031 HkLEQVQ4jdXtsa2EMAWGYCMX7sICkVqjXvACBe7CarASxdal1AWg54HmW5zEVs+mvSgS+ZBQ\
6032 8gcb4B4H7zvw8szMSaUBHm+KA4QC8LDpDn8ogT4UpPGci2j78IGFX3eLwPwAhknvYveQ\
6033 UEBDXaB0X2aJueYDOZnklQassPckjc4nW3E1Sfwqy6jU/vAKPhg0AL5Fhve84QdCWDMwz\
6034 yMGSsUPYWHMAr19k0tkV2sb3sdw2rUCqW88q4Rf1A9s1JPv9cTpnRD4XfkIn8XaQCiwT6Lzq\
6035 Z08dhw/4+U2Gzq1s8gbqVmkfr1N6YXR8OqLD00mlGTWvzPERA8AL9vbv0iFpSoL33fsVytrL\
6036 S9wiqDznhUI38v5n783/gBUUs2ELgic8GAAAABJR5ErkJggg==";
6037
6038 </script>
6039
6040 <script id="gsh-script">
6041 //document.getElementById('GshFaviconURL').href = GShellFavicon
6042 document.getElementById('GshFaviconURL').href = GShellInsideIcon
6043 //document.getElementById('GshFaviconURL').href = ITSmoreQR
6044 //document.getElementById('GshFaviconURL').href = GShellLogo
6045
6046 // id of GShell HTML elemets
6047 var E_BANNER = "GshBanner" // banner element in HTML
6048 var E_FOOTER = "GshFooter" // footer element in HTML
6049 var E_GINDEX = "gsh-gindex" // index of Golang code of GShell
6050 var E_GOCODE = "gsh-gocode" // Golang code of GShell
6051 var E_TODO = "gsh-todo" // TODO of GShell
6052 var E_DICT = "gsh-dict" // Dictionaly of GShell
6053
6054 function bannerElem(){ return document.getElementById(E_BANNER); }
6055 function bannerStyleFunc(){ return bannerElem().style; }
6056 var bannerStyle = bannerStyleFunc()
6057 bannerStyle.backgroundImage = "url("+GShellLogo+")";
6058 //bannerStyle.backgroundImage = "url("+GShellInsideIcon+")";
6059 //bannerStyle.backgroundImage = "url("+GShellFavicon+")";
6060 GMenu.style.backgroundImage = "url("+GShellInsideIcon+")";
6061
6062 function footerElem(){ return document.getElementById(E_FOOTER); }
6063 function footerStyle(){ return footerElem().style; }
6064 footerElem().style.backgroundImage="url("+ITSmoreQR+")";
6065 //footerStyle().backgroundImage = "url("+ITSmoreQR+")";
6066
6067 function html_fold(e){
6068   if( e.innerHTML == "Fold" ){
6069     e.innerHTML = "Unfold"
6070     document.getElementById('gsh-menu-exit').innerHTML=""
6071     document.getElementById('GshStatement').open=false
6072     GshFeatures.open = false
6073     document.getElementById('html-src').open=false
6074     document.getElementById(E_GINDEX).open=false
6075     document.getElementById(E_GOCODE).open=false
6076     document.getElementById(E_TODO).open=false
6077     document.getElementById('references').open=false
6078   }else{
6079     e.innerHTML = "Fold"
6080     document.getElementById('GshStatement').open=true
6081     GshFeatures.open = true
6082     document.getElementById(E_GINDEX).open=true
6083     document.getElementById(E_GOCODE).open=true
6084     document.getElementById(E_TODO).open=true
6085     document.getElementById('references').open=true
6086   }
6087 }
6088 function html_pure(e){
6089   if( e.innerHTML == "Pure" ){
6090     document.getElementById('gsh').style.display=true
6091     //document.style.display = false
6092     e.innerHTML = "Unpure"
6093   }else{
6094     document.getElementById('gsh').style.display=false
6095     //document.style.display = true
6096     e.innerHTML = "Pure"
6097   }
6098 }
6099
6100 var bannerIsStopping = false
6101 //NOTE: .com/JSRFP/prop_style_backgroundposition.asp
6102 function shiftBG(){
6103   bannerIsStopping = !bannerIsStopping
6104   bannerStyle.backgroundPosition = "0 0";
6105 }
6106 // status should be inherited on Window Fork(), so use the status in DOM
6107 function html_stop(e,toggle){
6108   if( toggle ){
6109     if( e.innerHTML == "Stop" ){
6110       bannerIsStopping = true
6111       e.innerHTML = "Start"
6112     }else{
6113       bannerIsStopping = false
6114       e.innerHTML = "Stop"
6115     }
6116   }else{
6117     // update JavaScript variable from DOM status
6118     if( e.innerHTML == "Stop" ){ // shown if it's running
6119       bannerIsStopping = false
6120     }else{
6121       bannerIsStopping = true
6122     }
6123   }
6124 }

```

```

6125 html_stop(document.getElementById('GshMenuStop'),false) // onInit.
6126 //html_stop(bannerElem(),false) // onInit.
6127
6128 //https://www.w3schools.com/jsref/met_win_setinterval.asp
6129 function shiftBanner(){
6130     var now = new Date().getTime();
6131     //console.log("now="+now%10)
6132     if( !bannerIsStopping ){
6133         bannerStyle.backgroundPosition = ((now/10)%100000)+" 0";
6134     }
6135 }
6136 setInterval(shiftBanner,10); // onInit.
6137
6138 // <a href="https://developer.mozilla.org/ja/docs/Web/API/Window/open">window.open(</a>
6139 // from embedded html to standalone page
6140 var MyChildren = 0
6141 function html_fork(){
6142     MyChildren += 1
6143     WinId = document.getElementById('gsh-WinId').innerHTML + "." + MyChildren;
6144     newwin = window.open("",WinId,"");
6145     src = document.getElementById("gsh");
6146     srctml = src.outerHTML;
6147     newwin.document.write("<"+srctml>");
6148     newwin.document.write(srctml);
6149     newwin.document.write("<"+"/html>");
6150     newwin.document.getElementById('gsh-menu-exit').innerHTML = "Close";
6151     newwin.document.getElementById('gsh-WinId').innerHTML = WinId;
6152     newwin.document.close();
6153     newwin.focus();
6154 }
6155 function html_close(){
6156     window.close()
6157 }
6158 function win_jump(win){
6159     //win = window.top;
6160     win = window.opener; // https://developer.mozilla.org/ja/docs/Web/API/window.opener
6161     if( win == null ){
6162         console.log("jump to window.opener("+win+") (Error)\n")
6163     }else{
6164         console.log("jump to window.opener("+win+")\n")
6165         win.focus();
6166     }
6167 }
6168
6169 // 0.2.9 2020-0902 created checksum of HTML
6170 CRC32UNIX = 0x04c11db7 // Unix cksum
6171 function byteCRC32add(bigcrc,octstr,octlen){
6172     var crc = new Int32Array(1)
6173     crc[0] = bigcrc
6174
6175     let oi = 0
6176     for( ; oi < octlen; oi++){
6177         var oct = new Int8Array(1)
6178         oct[0] = octstr[oi]
6179         for( bi = 0; bi < 8; bi++){
6180             //console.log("--CRC32 "+crc[0]+" "+oct[0].toString(16)+" ["+oi+"."+bi+"]\n")
6181             ovf1 = crc[0] < 0 ? 1 : 0
6182             ovf2 = oct[0] < 0 ? 1 : 0
6183             ovf = ovf1 ^ ovf2
6184             oct[0] <<= 1
6185             crc[0] <<= 1
6186             if( ovf ){ crc[0] ^= CRC32UNIX }
6187         }
6188     }
6189     //console.log("--CRC32 byteAdd return crc="+crc[0]+","+oi+"/"+octlen+"\n")
6190     return crc[0];
6191 }
6192 function strCRC32add(bigcrc,stri,strlen){
6193     var crc = new Uint32Array(1)
6194     crc[0] = bigcrc
6195     var code = new Uint8Array(strlen);
6196     for( i = 0; i < strlen; i++){
6197         code[i] = stri.charCodeAtAt(i) // not charAt() !!!!
6198         //console.log("=== "+code[i].toString(16)+" <<== "+stri[i]+"")
6199     }
6200     crc[0] = byteCRC32add(crc,code,strlen)
6201     //console.log("--CRC32 strAdd return crc="+crc[0]+"")
6202     return crc[0]
6203 }
6204 function byteCRC32end(bigcrc,len){
6205     var crc = new Uint32Array(1)
6206     crc[0] = bigcrc
6207     var slen = new Uint8Array(4)
6208     let li = 0
6209     for( ; li < 4; ){
6210         slen[li] = len
6211         li += 1
6212         len >>= 8
6213         if( len == 0 ){
6214             break
6215         }
6216     }
6217     crc[0] = byteCRC32add(crc[0],slen,li)
6218     crc[0] ^= 0xFFFFFFFF
6219     return crc[0]
6220 }
6221 function strCRC32(stri,len){
6222     var crc = new Uint32Array(1)
6223     crc[0] = 0
6224     crc[0] = strCRC32add(0,stri,len)
6225     crc[0] = byteCRC32end(crc[0],len)
6226     //console.log("--CRC32 "+crc[0]+" "+len+"\n")
6227     return crc[0]
6228 }
6229 function getSourceText(){
6230     version = document.getElementById('GshVersion').innerHTML
6231     sfavico = document.getElementById('GshFaviconURL').href;
6232     sbanner = document.getElementById('GshBanner').style.backgroundImage;
6233     spositi = document.getElementById('GshBanner').style.backgroundPosition;
6234     sfooter = document.getElementById('GshFooter').style.backgroundImage;
6235
6236     if( document.getElementById('GJC_1') ){ GJC_1.remove() }
6237
6238     // these should be removed by CSS selector or class, after sevaed to non-printed attribute
6239     GshBanner.removeAttribute('style');
6240     GshFooter.removeAttribute('style');
6241     document.getElementById('GshMenuSign').removeAttribute("style");
6242     styleGMenu = GMenu.getAttribute("style")
6243     GMenu.removeAttribute("style");
6244     styleGStat = GStat.getAttribute("style")
6245     GStat.removeAttribute("style");
6246     styleGTop = GTop.getAttribute("style")
6247     GTop.removeAttribute("style");
6248     styleGshGrid = GshGrid.getAttribute("style")
6249     GshGrid.removeAttribute("style");

```

```

6250 styleGPos = GPos.getAttribute("style");
6251 GPos.removeAttribute("style");
6252 GPos.innerHTML = "";
6253 styleGLog = GLog.getAttribute("style");
6254 GLog.removeAttribute("style");
6255 GLog.innerHTML = "";
6256 styleGShellPlane = GShellPlane.getAttribute("style")
6257 GShellPlane.removeAttribute("style")
6258 styleRawTextViewer = RawTextViewer.getAttribute("style")
6259 RawTextViewer.removeAttribute("style")
6260 styleRawTextViewerClose = RawTextViewerClose.getAttribute("style")
6261 RawTextViewerClose.removeAttribute("style")
6262
6263 GshFaviconURL.href = "";
6264
6265 //it seems that interHTML and outerHTML generate style="" for these (??)
6266 //GshBanner.removeAttribute('style');
6267 //GshFooter.removeAttribute('style');
6268 //GshMenuSign.removeAttribute('style');
6269 GshBanner.style=""
6270 GshFooter.style=""
6271 GshMenuSign.style=""
6272
6273 textarea = document.createElement("textarea")
6274 srchtml = document.getElementById("gsh").outerHTML;
6275 //textarea = document.createElement("textarea")
6276 // 2020-0910 ?? ... this causes inserting style="" to Banner and Footer,
6277 // with Chromium?/ after reloading from file:///
6278 textarea.innerHTML = srchtml
6279 // <a href="https://stackoverflow.com/questions/5796718/html-entity-decode">Thanks</a>
6280 var rawtext = textarea.value
6281 //textarea.destroy()
6282 //rawtext = gsh.textContent // this removes #include <FILENAME> too
6283 var orgtext = ""
6284 + "/*<+>html>\n" // lost preamble text
6285 + rawtext
6286 + "<+>/html>\n" // lost trail text
6287 ;
6288
6289 tlen = orgtext.length
6290 console.log("length="+tlen+"\n")
6291 document.getElementById('GshFaviconURL').href = sfavico;
6292
6293 document.getElementById('GshBanner').style.backgroundImage = sbanner;
6294 document.getElementById('GshBanner').style.backgroundPosition = spositi;
6295 document.getElementById('GshFooter').style.backgroundImage = sfooter;
6296
6297 GStat.setAttribute("style",styleGStat)
6298 GMenu.setAttribute("style",styleGMenu)
6299 CTop.setAttribute("style",styleGTop)
6300 GLog.setAttribute("style",styleGLog)
6301 GPos.setAttribute("style",styleGPos)
6302 GshGrid.setAttribute("style",styleGshGrid)
6303 GShellPlane.setAttribute("style",styleGShellPlane)
6304 RawTextViewer.setAttribute("style",styleRawTextViewer)
6305 RawTextViewerClose.setAttribute("style",styleRawTextViewerClose)
6306 return orgtext
6307 }
6308 function getDigest(){
6309 var text = ""
6310 text = getSourceText()
6311 var digest = ""
6312 tlen = text.length
6313 digest = strCRC32(text,tlen) + " " + tlen
6314 return { text, digest }
6315 }
6316 function html_digest(){
6317 version = document.getElementById('GshVersion').innerHTML
6318 let {text, digest} = getDigest()
6319 alert("cksum: " + digest + " " + version)
6320 }
6321 function charsIn(stri,char){
6322 ln = 0;
6323 for( i = 0; i < stri.length; i++){
6324 if( stri.charCodeAt(i) == char.charCodeAt(0) )
6325 ln++;
6326 }
6327 return ln;
6328 }
6329
6330 //class digestElement extends HTMLElement { }
6331 //< script>customElements.define('digest',digestElement)< /script>
6332 function showDigest(e){
6333 result = 'version=' + GshVersion.innerHTML + '\n'
6334 result += 'lines=' + e.dataset.lines + '\n'
6335 + 'length=' + e.dataset.length + '\n'
6336 + 'crc32u=' + e.dataset.crc32u + '\n'
6337 + 'time=' + e.dataset.time + '\n';
6338
6339 alert(result)
6340 }
6341
6342 function html_sign(e){
6343 if( RawTextViewer.style.zIndex == 1000 ){
6344 hideRawTextViewer()
6345 return
6346 }
6347 //gsh_digest_.innerHTML = "";
6348 text = getSourceText() // the original text
6349 tlen = text.length
6350 digest = strCRC32(text,tlen)
6351 //gsh_digest_.innerHTML = digest + " " + tlen
6352 //text = getSourceText() // the text with its digest
6353 Lines = charsIn(text,'\n')
6354
6355 name = "gsh"
6356 sid = name + "-digest"
6357 d = new Date()
6358 signedAt = d.getTime()
6359
6360 sign = '/'+'<'+span\n'
6361 + ' id="' + sid + '"\n'
6362 + ' class=" digest "\n'
6363 + ' data-target-id="'+name+'"\n'
6364 + ' data-crc32u="' + digest + '"\n'
6365 + ' data-length="' + tlen + '"\n'
6366 + ' data-lines="' + Lines + '"\n'
6367 + ' data-time="' + signedAt + '"\n'
6368 + '><' + '/span>\n*'+/\n'
6369
6370 text = sign + text
6371
6372 txhtml = '<' + 'table id="LineNumbered"><' + 'tr><' + 'td>'
6373 + '<' + 'textarea cols=5 rows=' + Lines + ' class="LineNumber">'
6374 for( i = 1; i <= Lines; i++){

```

```

6375     txthtml += i.toString() + '\n'
6376 }
6377 txthtml += ""
6378 + '<' + '/textarea>'
6379 + '<' + '/td><' + 'td>'
6380 + '<' + 'textarea cols=150 rows=' + Lines + 'spellcheck="false"'
6381 + ' class="LineNumbered">'
6382 + text + '<' + '/textarea>'
6383 + '<' + '/td><' + '/tr><' + '/table>'
6384
6385 for( i = 1; i <= 30; i++){
6386     txthtml += '<br>\n'
6387 }
6388 RawTextViewer.innerHTML = txthtml
6389
6390 btn = e
6391 e.style.color = "rgba(128,128,255,0.9)";
6392 y = e.getBoundingClientRect().top.toFixed(0)
6393 //h = e.getBoundingClientRect().height.toFixed(0)
6394 RawTextViewer.style.top = Number(y) + 30
6395 RawTextViewer.style.left = 100;
6396 RawTextViewer.style.height = window.innerHeight - 20;
6397 //RawTextViewer.style.opacity = 1.0;
6398 //RawTextViewer.style.backgroundColor = "rgba(0,0,0,0.0)";
6399 RawTextViewer.style.backgroundColor = "rgba(255,255,255,0.8)";
6400 RawTextViewer.style.zIndex = 1000;
6401 RawTextViewer.style.display = true;
6402
6403 if( RawTextViewerClose.style == null ){
6404     RawTextViewerClose.style = "";
6405 }
6406 RawTextViewerClose.style.top = Number(y) + 10
6407 RawTextViewerClose.style.left = 100;
6408 RawTextViewerClose.style.zIndex = 1001;
6409
6410 ScrollToElement(CurElement,RawTextViewerClose)
6411 }
6412 function hideRawTextViewer(){
6413     RawTextViewer.style.left = 10000;
6414     RawTextViewer.style.zIndex = -100;
6415     RawTextViewer.style.opacity = 0.0;
6416     RawTextViewer.style = null
6417     RawTextViewer.innerHTML = "";
6418
6419     GshMenuSign.style.color = "rgba(255,128,128,1.0)";
6420     RawTextViewerClose.style.top = 0;
6421     RawTextViewerClose.style = null
6422 }
6423
6424 // source code viewer
6425 function frame_close(){
6426     srcframe = document.getElementById("src-frame");
6427     srcframe.innerHTML = "";
6428     //srcframe.style.cols = 1;
6429     srcframe.style.rows = 1;
6430     srcframe.style.height = 0;
6431     srcframe.style.display = false;
6432     src = document.getElementById("src-frame-textarea");
6433     src.innerHTML = ""
6434     //src.cols = 0
6435     src.rows = 0
6436     src.display = false
6437     //alert("--closed--")
6438 }
6439 //<!-- | <span onclick="html_view();">Source</span> -->
6440 //<!-- | <span onclick="frame_close();">SourceClose</span> -->
6441 //<!--| <span>Download</span> -->
6442 function frame_open(){
6443     document.getElementById('GshFaviconURL').href = "";
6444     oldsrc = document.getElementById("GENSRC");
6445     if( oldsrc != null ){
6446         //alert("--I--(erasing old text)")
6447         oldsrc.innerHTML = "";
6448         return
6449     }else{
6450         //alert("--I--(no old text)")
6451     }
6452     styleBanner = GshBanner.getAttribute("style")
6453     GshBanner.removeAttribute("style")
6454     styleFooter = GshFooter.getAttribute("style")
6455     GshFooter.removeAttribute("style")
6456     if( document.getElementById('GJC_1') ){ GJC_1.remove() }
6457
6458     GshFaviconURL.href = "";
6459     GStat.removeAttribute('style')
6460     GshGrid.removeAttribute('style')
6461     GshMenuSign.removeAttribute('style')
6462     GPos.removeAttribute('style')
6463     GPos.innerHTML = "";
6464     GLog.removeAttribute('style')
6465     GLog.innerHTML = "";
6466     GMenu.removeAttribute('style')
6467     GTop.removeAttribute('style')
6468     GShellPlane.removeAttribute('style')
6469     RawTextViewer.removeAttribute('style')
6470     RawTextViewerClose.removeAttribute('style')
6471
6472     src = document.getElementById("gsh");
6473     srhtml = src.outerHTML
6474     srcframe = document.getElementById("src-frame");
6475     srcframe.innerHTML = ""
6476     + "<"+cite id="GENSRC">\n"
6477     + "<"+style>\n"
6478     + "#GENSRC textarea{tab-size:4;}\n"
6479     + "#GENSRC textarea{-o-tab-size:4;}\n"
6480     + "#GENSRC textarea{-moz-tab-size:4;}\n"
6481     + "#GENSRC textarea{spellcheck:false;}\n"
6482     + "<"+style>\n"
6483     + "<"+textarea id="src-frame-textarea" cols=100 rows=20 class="gsh-code">"
6484     + /*<"+html>\n" // lost preamble text
6485     + srhtml
6486     + "<"+html>\n" // lost trail text
6487     + "<"+textarea>\n"
6488     + "<"+cite>!-- GENSRC -->\n";
6489
6490     //srcframe.style.cols = 80;
6491     //srcframe.style.rows = 80;
6492
6493     GshBanner.setAttribute('style',styleBanner)
6494     GshFooter.setAttribute('style',styleFooter)
6495 }
6496 function fill_CSSView(){
6497     part = document.getElementById('GshStyleDef')
6498     view = document.getElementById('gsh-style-view')
6499     view.innerHTML = ""

```

```

6500 + "<"+'textarea cols=100 rows=20 class="gsh-code">'
6501 + part.innerHTML
6502 + "<"+'/textarea>"
6503 }
6504 function fill_JavaScriptView(){
6505     jspart = document.getElementById('gsh-script')
6506     view = document.getElementById('gsh-script-view')
6507     view.innerHTML = ""
6508     + "<"+'textarea cols=100 rows=20 class="gsh-code">'
6509     + jspart.innerHTML
6510     + "<"+'/textarea>"
6511 }
6512 function fill_DataView(){
6513     part = document.getElementById('gsh-data')
6514     view = document.getElementById('gsh-data-view')
6515     view.innerHTML = ""
6516     + "<"+'textarea cols=100 rows=20 class="gsh-code">'
6517     + part.innerHTML
6518     + "<"+'/textarea>"
6519 }
6520 function jumpto_StyleView(){
6521     jsview = document.getElementById('html-src')
6522     jsview.open = true
6523     jsview = document.getElementById('gsh-style-frame')
6524     jsview.open = true
6525     fill_CSSView()
6526 }
6527 function jumpto_JavaScriptView(){
6528     jsview = document.getElementById('html-src')
6529     jsview.open = true
6530     jsview = document.getElementById('gsh-script-frame')
6531     jsview.open = true
6532     fill_JavaScriptView()
6533 }
6534 function jumpto_DataView(){
6535     jsview = document.getElementById('html-src')
6536     jsview.open = true
6537     jsview = document.getElementById('gsh-data-frame')
6538     jsview.open = true
6539     fill_DataView()
6540 }
6541 function jumpto_WholeView(){
6542     jsview = document.getElementById('html-src')
6543     jsview.open = true
6544     jsview = document.getElementById('gsh-whole-view')
6545     jsview.open = true
6546     frame_open()
6547 }
6548 function html_view(){
6549     html_stop();
6550
6551     banner = document.getElementById('GshBanner').style.backgroundImage;
6552     footer = document.getElementById('GshFooter').style.backgroundImage;
6553     document.getElementById('GshBanner').style.backgroundImage = "";
6554     document.getElementById('GshBanner').style.backgroundColor = "";
6555     document.getElementById('GshFooter').style.backgroundImage = "";
6556
6557     //srcwin = window.open("", "CodeView2", "");
6558     srcwin = window.open("", "", "");
6559     srcwin.document.write("<span id='gsh'>\n");
6560
6561     src = document.getElementById("gsh");
6562     srcwin.document.write("<"+'style>\n");
6563     srcwin.document.write("textarea{tab-size:4;}\n");
6564     srcwin.document.write("textarea{-o-tab-size:4;}\n");
6565     srcwin.document.write("textarea{-moz-tab-size:4;}\n");
6566     srcwin.document.write("</style>\n");
6567     srcwin.document.write("<h2>\n");
6568     srcwin.document.write("<"+'span onclick='\"window.close();\">Close</span> | \n");
6569     //srcwin.document.write("<"+'span onclick='\"html_stop();\">Run</span>\n");
6570     srcwin.document.write("</h2>\n");
6571     srcwin.document.write("<textarea id='gsh-src-src' cols=100 rows=60>");
6572     srcwin.document.write("<"+'html>\n");
6573     srcwin.document.write("<"+'span id='\"gsh\">");
6574     srcwin.document.write(src.innerHTML);
6575     srcwin.document.write("<"+'/span><"+'/html>\n");
6576     srcwin.document.write("<"+'textarea>\n");
6577
6578     document.getElementById('GshBanner').style.backgroundImage = banner;
6579     document.getElementById('GshFooter').style.backgroundImage = footer
6580
6581     sty = document.getElementById("GshStyleDef");
6582     srcwin.document.write("<"+'style>\n");
6583     srcwin.document.write(sty.innerHTML);
6584     srcwin.document.write("<"+'/style>\n");
6585
6586     run = document.getElementById("gsh-script");
6587     srcwin.document.write("<"+'script>\n");
6588     srcwin.document.write(run.innerHTML);
6589     srcwin.document.write("<"+'/script>\n");
6590
6591     srcwin.document.write("<"+'/span><"+'/html>\n"); // gsh span
6592     srcwin.document.close();
6593     srcwin.focus();
6594 }
6595 GSH = document.getElementById("gsh")
6596
6597 //GSH.onclick = "alert('Ouch!');"
6598 //GSH.css = "{background-color:#eef;}";
6599 //GSH.style = "background-color:#eef;";
6600 //GSH.style.display = false;
6601 //alert('Ouch0!')
6602 //GSH.style.display = true;
6603
6604 // 2020-0904 created, tentative
6605 document.addEventListener('keydown', jgshCommand);
6606 //CurElement = GshStatement
6607 CurElement = GshMenu
6608 MemElement = GshMenu
6609
6610 function nextSib(e){
6611     n = e.nextSibling;
6612     for( i = 0; i < 100; i++){
6613         if( n == null ){
6614             break;
6615         }
6616         if( n.nodeName == "DETAILS" ){
6617             return n;
6618         }
6619         n = n.nextSibling;
6620     }
6621     return null;
6622 }
6623 function prevSib(e){
6624     n = e.previousSibling;

```

```

6625     for( i = 0; i < 100; i++){
6626         if( n == null ){
6627             break;
6628         }
6629         if( n.nodeName == "DETAILS" ){
6630             return n;
6631         }
6632         n = n.previousSibling;
6633     }
6634     return null;
6635 }
6636 function setColor(e,eName,eColor){
6637     if( e.hasChildNodes() ){
6638         s = e.childNodes;
6639         if( s != null ){
6640             for( ci = 0; ci < s.length; ci++ ){
6641                 if( s[ci].nodeName == eName ){
6642                     s[ci].style.color = eColor;
6643                     //s[ci].style.backgroundColor = eColor;
6644                     break;
6645                 }
6646             }
6647         }
6648     }
6649 }
6650
6651 // https://docs.microsoft.com/en-us/previous-versions//hh781509(v=vs.85)
6652 function showCurElementPosition(ev){
6653     if( document.getElementById("GPos") == null ){
6654         return;
6655     }
6656     if( GPos == null ){
6657         return;
6658     }
6659     e = CurElement
6660     y = e.getBoundingClientRect().top.toFixed(0)
6661     x = e.getBoundingClientRect().left.toFixed(0)
6662
6663     h = ev + " "
6664     h += 'y='+y+", "+ 'x='+x+" -- "
6665     h += "w="+ window.innerWidth + ", h=" + window.innerHeight + " -- "
6666     //GPos.test = h
6667     //GPos.innerHTML = h
6668     GPos.innerHTML = h
6669 }
6670
6671 function DateLong(){
6672     d = new Date()
6673     return d.getFullYear() + "/" + d.getMonth() + "/" + d.getDate() + " "
6674     + d.getHours() + ":" + d.getMinutes() + ":" + d.getSeconds()
6675     + "." + d.getMilliseconds()
6676     + " " + d.getTimezoneOffset()/60
6677     + " " + d.getTime() + "." + d.getMilliseconds()
6678 }
6679
6680
6681 function GShellMenu(e){
6682     GLog.innerHTML = "Hello, World! (" + DateLong() + ")"
6683     showGShellPlane()
6684 }
6685 // placements of planes
6686 function GShellResizeX(ev){
6687     //if( document.getElementById("GMenu") != null ){
6688         GMenu.style.left = window.innerWidth - 100
6689         GMenu.style.top = window.innerHeight - 90
6690         console.log("place GMENU "+GMenu.style.left+" "+GMenu.style.top)
6691     }
6692     //}
6693     GStat.style.width = window.innerWidth
6694     //if( document.getElementById("GPos") != null ){
6695         GPos.style.width = window.innerWidth
6696         GPos.style.top = window.innerHeight - 30; //GPos.style.height
6697     //}
6698     if( document.getElementById("GLog") != null ){
6699         GLog.style.width = window.innerWidth
6700         GLog.innerHTML = ""
6701     }
6702     if( document.getElementById("GLog") != null ){
6703         //GLog.innerHTML = "Resize: w=" + window.innerWidth +
6704         //", h=" + window.innerHeight
6705     }
6706     showCurElementPosition(ev)
6707 }
6708 function GShellResize(){
6709     GShellResizeX("RESIZE")
6710 }
6711 window.onresize = GShellResize
6712
6713 function ScrollToElement(oe,ne){
6714     ne.scrollIntoView()
6715     ny = ne.getBoundingClientRect().top.toFixed(0)
6716     nx = ne.getBoundingClientRect().left.toFixed(0)
6717     //GLog.innerHTML = "["+ny+","+nx+"]"
6718     //window.scrollTo(0,0)
6719
6720     GTop.style.backgroundColor = "rgba(0,0,0,0.0)"
6721     GshGrid.style.left = '250px';
6722     GshGrid.style.zIndex = 0
6723     return
6724     oy = oe.getBoundingClientRect().top.toFixed(0)
6725     ox = oe.getBoundingClientRect().left.toFixed(0)
6726     y = e.getBoundingClientRect().top.toFixed(0)
6727     x = e.getBoundingClientRect().left.toFixed(0)
6728     window.scrollTo(x,y)
6729     ny = e.getBoundingClientRect().top.toFixed(0)
6730     nx = e.getBoundingClientRect().left.toFixed(0)
6731     GLog.innerHTML = "["+oy+","+ox+"]->["+y+","+x+"]->["+ny+","+nx+"]"
6732 }
6733 var MyHistory = ""
6734 function showGShellPlane(){
6735     if( GShellPlane.style.zIndex == 0 ){
6736         GShellPlane.style.zIndex = 1000;
6737         GShellPlane.style.left = 30;
6738         GShellPlane.style.height = 320;
6739         GShellPlane.innerHTML = DateLong() + "<br>" +
6740         "-- History --<br>" + MyHistory;
6741     }else{
6742         GShellPlane.style.zIndex = 0;
6743         GShellPlane.style.left = 0;
6744         GShellPlane.style.height = 50;
6745         GShellPlane.innerHTML = "";
6746     }
6747 }
6748 var SuppressGJShell = false
6749 function jgshCommand(event){

```

```

6750 if( SuppressGJShell ){
6751     return
6752 }
6753 key = event
6754 keycode = key.code
6755 //GStat.style.width = window.innerWidth
6756 GStat.style.backgroundColor = "rgba(0,0,0,0.4)"
6757
6758 console.log("JSGsh-Key:"+keycode+"(^-^)/")
6759 if( keycode == "Digit0" ){ // fold side-bar
6760     // "Zero page"
6761     showGShellPlane();
6762 }else
6763 if( keycode == "Digit1" ){ // fold side-bar
6764     primary.style.width = "94%"
6765     secondary.style.width = "0%"
6766     secondary.style.opacity = 0
6767     GStat.innerHTML = "[Single Column View]"
6768 }else
6769 if( keycode == "Digit2" ){ // unfold side-bar
6770     primary.style.width = "58%"
6771     secondary.style.width = "36%"
6772     secondary.style.opacity = 1
6773     GStat.innerHTML = "[Double Column View]"
6774 }else
6775 if( keycode == "KeyU" ){ // fold/unfold all
6776     html_fold(GshMenuFold);
6777     location.href = "#"+CurElement.id;
6778 }else
6779 if( keycode == "KeyO" || keycode == "ArrowRight" ){ // fold the element
6780     CurElement.open = !CurElement.open;
6781 }else
6782 if( keycode == "ArrowRight" ){ // unfold the element
6783     CurElement.open = true
6784 }else
6785 if( keycode == "ArrowLeft" ){ // unfold the element
6786     CurElement.open = false
6787 }else
6788 if( keycode == "KeyI" ){ // inspect the element
6789     e = CurElement
6790     GLog.innerHTML = "Current Element: " + e + "<br>"
6791     + "name="+e.nodeName + ", "
6792     + "id="+e.id + " "
6793     + "children="+e.childNodes.length + ", "
6794     + "parent="+e.parentNode.id + "<br>"
6795     + "text="+e.textContent
6796     GStat.style.backgroundColor = "rgba(0,0,0,0.8)"
6797     return
6798 }else
6799 if( keycode == "KeyM" ){ // memory the position
6800     MemElement = CurElement
6801 }else
6802 if( keycode == "KeyN" || keycode == "ArrowDown" ){ // next element
6803     e = nextSib(CurElement)
6804     if( e != null ){
6805         setColor(CurElement,"SUMMARY","#fff")
6806         setColor(e,"SUMMARY","#8f8") // should be complement ?
6807         oe = CurElement
6808         CurElement = e
6809         //location.href = "#"+e.id;
6810         ScrollToElement(oe,e)
6811     }
6812 }else
6813 if( keycode == "KeyP" || keycode == "ArrowUp" ){ // previous element
6814     oe = CurElement
6815     e = prevSib(CurElement)
6816     if( e != null ){
6817         setColor(CurElement,"SUMMARY","#fff")
6818         setColor(e,"SUMMARY","#8f8") // should be complement ?
6819         CurElement = e
6820         //location.href = "#"+e.id;
6821         ScrollToElement(oe,e)
6822     }else{
6823         e = document.getElementById("GshBanner")
6824         if( e != null ){
6825             setColor(CurElement,"SUMMARY","#fff")
6826             CurElement = e
6827             ScrollToElement(oe,e)
6828         }else{
6829             e = document.getElementById("primary")
6830             if( e != null ){
6831                 setColor(CurElement,"SUMMARY","#fff")
6832                 CurElement = e
6833                 ScrollToElement(oe,e)
6834             }
6835         }
6836     }
6837 }else
6838 if( keycode == "KeyR" ){
6839     location.reload()
6840 }else
6841 if( keycode == "KeyJ" ){
6842     GshGrid.style.top = '120px';
6843     GshGrid.innerHTML = '(>_<){Down}';
6844 }else
6845 if( keycode == "KeyK" ){
6846     GshGrid.style.top = '0px';
6847     GshGrid.innerHTML = '^_^){Up}';
6848 }else
6849 if( keycode == "KeyH" ){
6850     GshGrid.style.left = '0px';
6851     GshGrid.innerHTML = '(_){Left}';
6852 }else
6853 if( keycode == "KeyL" ){
6854     GLog.innerHTML +=
6855     'screen='+screen.width+'px'+<br>'+
6856     'window='+window.innerWidth+'px'+<br>'+
6857     GshGrid.style.left = (document.documentElement.clientWidth-160).toString(10)+'px';
6858     GshGrid.innerHTML = '@_{Right}';
6859 }else
6860 if( keycode == "KeyS" ){
6861     html_stop(GshMenuStop,true)
6862 }else
6863 if( keycode == "KeyF" ){
6864     html_fork()
6865 }else
6866 if( keycode == "KeyC" ){
6867     window.close()
6868 }else
6869 if( keycode == "KeyD" ){
6870     html_digest()
6871 }else
6872 if( keycode == "KeyV" ){
6873     e = document.getElementById('gsh-digest')
6874     if( e != null ){

```

```

6875     showDigest(e)
6876     }
6877     }
6878
6879     showCurElementPosition("[ "+key.code+" ] --");
6880     if( document.getElementById("GPos") != null ){
6881         //GPos.innerHTML += "[ "+key.code+" ] --"
6882     }
6883     //GShellResizeX("[ "+key.code+" ] --");
6884 }
6885 GShellResizeX("INIT");
6886
6887 DisplaySize = "-- Display: "
6888 + 'screen='+screen.width+'px'
6889 + ', '+window='+window.innerWidth+'px';
6890
6891 // 2020-0909 added, permanent local storage
6892 // https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage
6893 Permanent = localStorage;
6894 MyHistory = Permanent.getItem('MyHistory')
6895 if( MyHistory == null ){ MyHistory = "" }
6896 d = new Date()
6897 MyHistory = d.getTime()/1000+" "+document.URL+"<br>" + MyHistory
6898 Permanent.setItem('MyHistory',MyHistory)
6899
6900 //Permanent.setItem('MyWindow',window)
6901 let {text, digest} = getDigest()
6902 GLog.innerHTML +=
6903     "-- GShell: " + GshVersion.innerHTML
6904     + "<br>" + "-- Digest: " + digest
6905     //+ "<br>" + DisplaySize
6906     //+ "<br>" + "-- LastVisit:<br>" + MyHistory
6907 GShellResizeX(null);
6908
6909 // <a href="https://www.w3.org/TR/WebCryptoAPI/">Web Cryptography API</a>
6910 //Convert a string into an ArrayBuffer
6911 //from https://developers.google.com/web/updates/2012/06/How-to-convert-ArrayBuffer-to-and-from-String
6912 function str2ab(str) {
6913     const buf = new ArrayBuffer(str.length);
6914     const bufView = new Uint8Array(buf);
6915     for (let i = 0, strLen = str.length; i < strLen; i++) {
6916         bufView[i] = str.charCodeAt(i);
6917     }
6918     return buf;
6919 }
6920 function importPrivateKey(pem) {
6921     const binaryDerString = window.atob(pemContents);
6922     const binaryDer = str2ab(binaryDerString);
6923     return window.crypto.subtle.importKey(
6924         "pkcs8",
6925         binaryDer,
6926         {
6927             name: "RSA-PSS",
6928             modulusLength: 2048,
6929             publicExponent: new Uint8Array([1, 0, 1]),
6930             hash: "SHA-256",
6931         },
6932         true,
6933         ["sign"]
6934     );
6935 }
6936 //importPrivateKey(ppem)
6937
6938 //key = {}
6939 //buf = "abc"
6940 //enc = "xyzxxxxxx"; //crypto.publicEncrypt(key,buf)
6941 //b64 = btoa(enc)
6942 //dec = atob(b64)
6943 //GLog.innerHTML = "enc:" + b64 + ", dec:" + dec
6944
6945 </script>
6946
6947 <span id="gjc" data-title="GJConsole" data-author="sato@its-more.jp">
6948 <!-- ----- GJConsole BEGIN { ----- -->
6949 <style id="GJConsoleStyle">
6950 .GJConsole {
6951     z-index:1000;
6952     width:500px; height:240px;
6953     color:#fff; background-color:#000;
6954     font-size:14px; font-family:monospace,Courier New;
6955 }
6956 </style>
6957
6958 <script id="GJConsoleScript" class="GJConsole">
6959 var PS1 = "§ "
6960 function GJC_KeyDown(keyevent){
6961     key = keyevent.code
6962     if( key == "Enter" ){
6963         GJC_Command(this)
6964         this.value += "\n" + PS1 // prompt
6965     }else
6966     if( key == "Escape"){
6967         SuppressGJShell = false
6968         GshMenu.focus() // should be previous focus
6969     }
6970 }
6971 function GJC_Command(text){
6972     lines = text.value.split('\n')
6973     line = lines[lines.length-1]
6974     argv = line.split(' ')
6975     text.value += '\n'
6976     if( argv[0] == '§' ){ argv.shift() }
6977     cmd = argv[0]
6978     argv.shift()
6979     args = argv.join(' ')
6980     if( cmd == 'echo' ){
6981         text.value += args
6982     }else
6983     if( cmd == 'date' ){
6984         text.value += DateLong()
6985     }else
6986     if( cmd == 'fork' ){
6987         html_fork()
6988     }else{
6989         text.value += "Commands: help | echo | date | fork | nife"
6990     }
6991 }
6992
6993 function GJC_Input(){
6994     if( this.value.endsWith("\n") ){ // remove NL added by textarea
6995         this.value = this.value.slice(0,this.value.length-1)
6996     }
6997 }
6998
6999 function GJC_FocusIn(){

```



```
7000     this.spellcheck = false
7001     SuppressGJShell = true
7002     this.onkeydown = GJC_Keydown
7003     this.style.zIndex = 20000
7004     this.style.backgroundColor = "rgba(0,0,0,1.0)"
7005     this.style.color = "rgba(255,255,1.0)"
7006 }
7007 function GJC_FocusOut(){
7008     SuppressGJShell = false
7009     this.removeEventListener('keydown',GJC_Keydown);
7010 }
7011 function GJC_Setup(gjcId){
7012     gjcId.value = "GJShell Console // " + GshVersion.innerHTML + "\n"
7013     //gjcId.value += "Date: " + DateLong() + "\n"
7014     gjcId.value += PS1
7015     gjcId.onfocus = GJC_FocusIn
7016     gjcId.addEventListener('input',GJC_Input);
7017     gjcId.addEventListener('focusout',GJC_FocusOut);
7018 }
7019 function GJC_Clear(id){
7020 }
7021 if( document.getElementById("GJC_0") != null ){
7022     GJC_Setup(GJC_0)
7023 }else{
7024     document.write('<'+ 'textarea id="GJC_1" class="GJConsole"><'+ '/textarea>')
7025     GJC_Setup(GJC_1)
7026 }
7027
7028 // TODO: focus handling
7029 </script>
7030 <!-- ----- GJConsole END } ----- -->
7031 </span>
7032
7033 *//<br></span></html>
7034
```