```
1  //<html>
2  /*<head>
3  <link rel=icon href=GShell-Logo05icon.png>
4  <meta charset=UTF-8>
5  <meta name="viewport" content="width=device-width, initial-scale=1.0">
6  <title>GShell-0.1.3 by SatoxITS</title>
7  </head>
8  <span id=gsh>
9  <header id=banner height=100px onclick="shiftBG();">
10 <div align=right><note>GShell version 0.1.3 // 2020-08-17 // SatoxITS</note></div>
11 </header>
12 <h2>GShell // a General purpose Shell built on the top of Golang</h2>
13 */
14 /*
15 <details id=overview><summary>Overview</summary><pre>
16 To be written
17 </pre></details>
18 */
19 /*
20 <details id=index><summary>Index</summary>
21 <pre onclick="document.getElementById('gocode').open=true;">
22
23 Implementation
24     Structures
25         <a href=#import>import</a>
26         <a href=#struct>struct</a>
27     Main functions
28         <a href=#comexpansion>str-expansion</a> // macro processor
29         <a href=#finder>finder</a>      // builtin find + du
30         <a href=#grep>grep</a>      // builtin grep + wc + cksum + ...
31         <a href=#plugin>plugin</a>      // plugin commands
32         <a href=#ex-commands>system</a>     // external commands
33         <a href=#builtin>builtin</a>        // builtin commands
34         <a href=#network>network</a>        // socket handler
35         <a href=#redirect>redirect</a>  // StdIn/Out redireciton
36         <a href=#history>history</a>        // command history
37         <a href=#rusage>rusage</a>      // resouce usage
38         <a href=#encode>encode</a>      // encode / decode
39         <a href=#getline>getline</a>        // line editor
40         <a href=#interpreter>interpreter</a>    // command interpreter
41         <a href=#main>main</a><pre>
42 </details>
43 */
44 //<details id=gocode><summary>Source Code</summary>
45 //<pre onclick="document.getElementById('gocode').open=false;">
46 // gsh - Go lang based Shell
47 // (c) 2020 ITS more Co., Ltd.
48 // 2020-0807 created by SatoxITS (sato@its-more.jp)
49
50 package main // gsh main
51 // <a name=import>Imported packages</a> // <a href=https://golang.org/pkg/>Packages</a>
52 import (
53     "fmt"       // <a href=https://golang.org/pkg/fmt/>fmt</a>
54     "strings"   // <a href=https://golang.org/pkg/strings/>strings</a>
55     "strconv"   // <a href=https://golang.org/pkg/strconv/>strconv</a>
56     "sort"      // <a href=https://golang.org/pkg/sort/>sort</a>
57     "time"      // <a href=https://golang.org/pkg/time/>time</a>
58     "bufio"     // <a href=https://golang.org/pkg/bufio/>bufio</a>
59     "io/ioutil" // <a href=https://golang.org/pkg/io/ioutil/>ioutil</a>
60     "os"        // <a href=https://golang.org/pkg/os/>os</a>
61     "syscall"   // <a href=https://golang.org/pkg/syscall/>syscall</a>
62     "plugin"    // <a href=https://golang.org/pkg/plugin/>plugin</a>
63     "net"       // <a href=https://golang.org/pkg/net/>net</a>
64     "net/http"  // <a href=https://golang.org/pkg/net/http/>http</a>
65     //"html"    // <a href=https://golang.org/pkg/html/>html</a>
66     "path/filepath" // <a href=https://golang.org/pkg/path/filepath/>filepath</a>
67     "go/types"  // <a href=https://golang.org/pkg/go/types/>types</a>
68     "go/token"  // <a href=https://golang.org/pkg/go/token/>token</a>
69     "encoding/base64"   // <a href=https://golang.org/pkg/encoding/base64/>base64</a>
70     //"gshdata" // gshell's logo and source code
71 )
72
73 var NAME = "gsh"
74 var VERSION = "0.1.3"
75 var DATE = "2020-0817b"
76 var LINESIZE = (8*1024)
77 var PATHSEP = ":"   // should be ";" in Windows
78 var DIRSEP = "/"    // canbe \ in Windows
79 var GSH_HOME = ".gsh"   // under home directory
80 var PROMPT = "> "
81
82 // -xX logging control
83 // --A-- all
84 // --I-- info.
85 // --D-- debug
86 // --W-- warning
87 // --E-- error
88 // --F-- fatal error
89
90 // <a name=struct>Structures</a>
91 type GCommandHistory struct {
92     StartAt     time.Time // command line execution started at
93     EndAt       time.Time // command line execution ended at
94     ResCode     int       // exit code of (external command)
95     CmdError    error     // error string
96     OutData     *os.File  // output of the command
97     FoundFile   []string  // output - result of ufind
98     Rusagev     [2]syscall.Rusage // Resource consumption, CPU time or so
99     CmdId       int       // maybe with identified with arguments or impact
100                          // redireciton commands should not be the CmdId
101    WorkDir     string    // working directory at start
102    CmdLine     string    // command line
103 }
104 type GChdirHistory struct {
105     Dir     string
106     MovedAt     time.Time
107 }
108 type CmdMode struct {
109     BackGround  bool
110 }
111 type PluginInfo struct {
112     Spec        *plugin.Plugin
113     Addr        plugin.Symbol
114     Name        string // maybe relative
115     Path        string // this is in Plugin but hidden
116 }
117 type GshContext struct {
118     StartDir    string  // the current directory at the start
119     GetLine     string  // gsh-getline command as a input line editor
120     ChdirHistory    []GChdirHistory // the 1st entry is wd at the start
121     gshPA       syscall.ProcAttr
122     CommandHistory  []GCommandHistory
123     CmdCurrent  GCommandHistory
124     BackGround  bool
125     BackGroundJobs  []int
126     LastRusage  syscall.Rusage
127     GshHomeDir  string
128     TerminalId  int
129     CmdTrace    bool
130     PluginFuncs []PluginInfo
131 }
132
133 func strBegins(str, pat string)(bool){
134     if 0 < len(str){
135         yes := str[0:len(pat)] == pat
```

```
        //fmt.Printf("--D-- strBegins(%v,%v)=%v\n",str,pat,yes)
        return yes
    }
    //fmt.Printf("--D-- strBegins(%v,%v)=%v\n",str,pat,false)
    return false
}
func isin(what string, list []string) bool {
    for _, v := range list  {
        if v == what {
            return true
        }
    }
    return false
}
func isinX(what string,list[]string)(int){
    for i,v := range list {
        if v == what {
            return i
        }
    }
    return -1
}

func env(opts []string) {
    env := os.Environ()
    if isin("-s", opts){
        sort.Slice(env, func(i,j int) bool {
            return env[i] < env[j]
        })
    }
    for _, v := range env {
        fmt.Printf("%v\n",v)
    }
}

// - rewriting should be context dependent
// - should postpone until the real point of evaluation
// - should rewrite only known notation of symobl
func scanInt(str string)(val int,leng int){
    leng = -1
    for i,ch := range str {
        if '0' <= ch && ch <= '9' {
            leng = i+1
        }else{
            break
        }
    }
    if 0 < leng {
        ival,_ := strconv.Atoi(str[0:leng])
        return ival,leng
    }else{
        return 0,0
    }
}
func substHistory(gshCtx *GshContext,str string,i int,rstr string)(leng int,rst string){
    if len(str[i+1:]) == 0 {
        return 0,rstr
    }
    hi := 0
    histlen := len(gshCtx.CommandHistory)
    if str[i+1] == '!' {
        hi = histlen - 1
        leng = 1
    }else{
        hi,leng = scanInt(str[i+1:])
        if leng == 0 {
            return 0,rstr
        }
        if hi < 0 {
            hi = histlen + hi
        }
    }
    if 0 <= hi && hi < histlen {
        //fmt.Printf("--D-- %v(%c)\n",str[i+leng:],str[i+leng])
        if 1 < len(str[i+leng:]) && str[i+leng:][1] == 'f' {
            leng += 1
            xlist := []string{}
            list := gshCtx.CommandHistory[hi].FoundFile
            for _,v := range list {
                //list[i] = escapeWhiteSP(v)
                xlist = append(xlist,escapeWhiteSP(v))
            }
            //rstr += strings.Join(list," ")
            rstr += strings.Join(xlist," ")
        }else{
            rstr += gshCtx.CommandHistory[hi].CmdLine
        }
    }else{
        leng = 0
    }
    return leng,rstr
}
func escapeWhiteSP(str string)(string){
    if len(str) == 0 {
        return "\\z" // empty, to be ignored
    }
    rstr := ""
    for _,ch := range str {
        switch ch {
            case '\\': rstr += "\\\\"
            case ' ': rstr += "\\s"
            case '\t': rstr += "\\t"
            case '\r': rstr += "\\r"
            case '\n': rstr += "\\n"
            default: rstr += string(ch)
        }
    }
    return rstr
}
func unescapeWhiteSP(str string)(string){ // strip original escapes
    rstr := ""
    for i := 0; i < len(str); i++ {
        ch := str[i]
        if ch == '\\' {
            if i+1 < len(str) {
                switch str[i+1] {
                    case 'z':
                        continue;
                }
            }
        }
        rstr += string(ch)
    }
    return rstr
}
func unescapeWhiteSPV(strv []string)([]string){ // strip original escapes
    ustrv := []string{}
    for _,v := range strv {
        ustrv = append(ustrv,unescapeWhiteSP(v))
    }
    return ustrv
}

// <a name=comexpansion>str-expansion</a>
// - this should be a macro processor
```

```
func strsubst(gshCtx *GshContext,str string,histonly bool) string {
    rstr := ""
    inEsc := 0 // escape characer mode
    for i := 0; i < len(str); i++ {
        //fmt.Printf("--D--Subst %v:%v\n",i,str[i:])
        ch := str[i]
        if inEsc == 0 {
            if ch == '!' {
                leng,xrstr := substHistory(gshCtx,str,i,rstr)
                if 0 < leng {
                    i += leng
                    rstr = xrstr
                    continue
                }
            }
            switch ch {
                case '\\': inEsc = '\\'; continue
                case '%':  inEsc = '%';  continue
                case '$':
            }
        }
        switch inEsc {
        case '\\':
            switch ch {
                case '\\': ch = '\\'
                case 's': ch = ' '
                case 't': ch = '\t'
                case 'r': ch = '\r'
                case 'n': ch = '\n'
                case 'z': inEsc = 0; continue // empty, to be ignored
            }
            inEsc = 0
        case '%':
            switch {
                case ch == '%': ch = '%'
                case ch == 'T':
                    rstr = rstr + time.Now().Format(time.Stamp)
                    continue;
                default:
                    // postpone the interpretation
                    rstr = rstr + "%" + string(ch)
                    continue;
            }
            inEsc = 0
        }
        rstr = rstr + string(ch)
    }
    return rstr
}
func showFileInfo(path string, opts []string) {
    if isin("-l",opts) || isin("-ls",opts) {
        fi, _ := os.Stat(path)
        mod := fi.ModTime()
        date := mod.Format(time.Stamp)
        fmt.Printf("%v %8v %s ",fi.Mode(),fi.Size(),date)
    }
    fmt.Printf("%s",path)
    if isin("-sp",opts) {
        fmt.Printf(" ")
    }else
    if ! isin("-n",opts) {
        fmt.Printf("\n")
    }
}
func userHomeDir()(string,bool){
    /*
    homedir,_ = os.UserHomeDir() // not implemented in older Golang
    */
    homedir,found := os.LookupEnv("HOME")
    //fmt.Printf("--I-- HOME=%v(%v)\n",homedir,found)
    if !found {
        return "/tmp",found
    }
    return homedir,found
}

func toFullpath(path string) (fullpath string) {
    if path[0] == '/' {
        return path
    }
    pathv := strings.Split(path,DIRSEP)
    switch {
    case pathv[0] == ".":
        pathv[0], _ = os.Getwd()
    case pathv[0] == "..": // all ones should be interpreted
        cwd, _ := os.Getwd()
        ppathv := strings.Split(cwd,DIRSEP)
        pathv[0] = strings.Join(ppathv,DIRSEP)
    case pathv[0] == "~":
        pathv[0],_ = userHomeDir()
    default:
        cwd, _ := os.Getwd()
        pathv[0] = cwd + DIRSEP + pathv[0]
    }
    return strings.Join(pathv,DIRSEP)
}

func IsRegFile(path string)(bool){
    fi, err := os.Stat(path)
    if err == nil {
        fm := fi.Mode()
        return fm.IsRegular();
    }
    return false
}

// <a name=encode>Encode / Decode</a>
// <a href=https://golang.org/pkg/encoding/base64/#example_NewEncoder>Encoder</a>
func Enc(gshCtx *GshContext,argv[]string)(*GshContext){
    file := os.Stdin
    buff := make([]byte,LINESIZE)
    li := 0
    encoder := base64.NewEncoder(base64.StdEncoding,os.Stdout)
    for li = 0; ; li++ {
        count, err := file.Read(buff)
        if count <= 0 {
            break
        }
        if err != nil {
            break
        }
        encoder.Write(buff[0:count])
    }
    encoder.Close()
    return gshCtx
}
func Dec(gshCtx *GshContext,argv[]string)(*GshContext){
    decoder := base64.NewDecoder(base64.StdEncoding,os.Stdin)
    li := 0
    buff := make([]byte,LINESIZE)
    for li = 0; ; li++ {
        count, err := decoder.Read(buff)
        if count <= 0 {
            break
        }
```

```
        if err != nil {
            break
        }
        os.Stdout.Write(buff[0:count])
    }
    return gshCtx
}
// lnsp [N] [-crlf][-C \\]
func SplitLine(gshCtx *GshContext,argv[]string)(*GshContext){
    reader := bufio.NewReaderSize(os.Stdin,64*1024)
    ni := 0
    toi := 0
    for ni = 0; ; ni++ {
        line, err := reader.ReadString('\n')
        if len(line) <= 0 {
            if err != nil {
            fmt.Fprintf(os.Stderr,"--I-- lnsp %d to %d (%v)\n",ni,toi,err)
            break
            }
        }
        off := 0
        ilen := len(line)
        remlen := len(line)
        for oi := 0; 0 < remlen; oi++ {
            olen := remlen
            addnl := false
            if 72 < olen {
                olen = 72
                addnl = true
            }
            fmt.Fprintf(os.Stderr,"--D-- write %d [%d.%d] %d %d/%d/%d\n",
                toi,ni,oi,off,olen,remlen,ilen)
            toi += 1
            os.Stdout.Write([]byte(line[0:olen]))
            if addnl {
                //os.Stdout.Write([]byte("\r\n"))
                os.Stdout.Write([]byte("\n"))
            }
            line = line[olen:]
            off += olen
            remlen -= olen
        }
    }
    fmt.Fprintf(os.Stderr,"--I-- lnsp %d to %d\n",ni,toi)
    return gshCtx
}

// <a name=grep>grep</a>
// "lines", "lin" or "lnp" for "(text) line processor" or "scanner"
// a*,!ab,c, ... sequentioal combination of patterns
// what "LINE" is should be definable
// generic line-by-line processing
// grep [-v]
// cat -n -v
// uniq [-c]
// tail -f
// sed s/x/y/ or awk
// grep with line count like wc
// rewrite contents if specified
func xGrep(gshCtx GshContext,path string,rexpv[]string)(int){
    file, err := os.OpenFile(path,os.O_RDONLY,0)
    if err != nil {
        fmt.Printf("--E-- grep %v (%v)\n",path,err)
        return -1
    }
    defer file.Close()
    if gshCtx.CmdTrace { fmt.Printf("--I-- grep %v %v\n",path,rexpv) }
    //reader := bufio.NewReaderSize(file,LINESIZE)
    reader := bufio.NewReaderSize(file,80)
    li := 0
    found := 0
    for li = 0; ; li++ {
        line, err := reader.ReadString('\n')
        if len(line) <= 0 {
            break
        }
        if 150 < len(line) {
            // maybe binary
            break;
        }
        if err != nil {
            break
        }
        if 0 <= strings.Index(string(line),rexpv[0]) {
            found += 1
            fmt.Printf("%s:%d: %s",path,li,line)
        }
    }
        //fmt.Printf("total %d lines %s\n",li,path)
    //if( 0 < found ){ fmt.Printf("((found %d lines %s))\n",found,path); }
    return found
}

// <a name=finder>Finder</a>
// finding files with it name and contents
// file names are ORed
// show the content with %x fmt list
// ls -R
// tar command by adding output
type fileSum struct {
    Err int64   // access error or so
    Size    int64   // content size
    DupSize int64   // content size from hard links
    Blocks  int64   // number of blocks (of 512 bytes)
    DupBlocks int64 // Blocks pointed from hard links
    HLinks  int64   // hard links
    Words   int64
    Lines   int64
    Files   int64
    Dirs    int64   // the num. of directories
    SymLink int64
    Flats   int64   // the num. of flat files
    MaxDepth    int64
    MaxNamlen   int64   // max. name length
    nextRepo    time.Time
}
func showFusage(dir string,fusage *fileSum){
    bsume := float64(((fusage.Blocks-fusage.DupBlocks)/2)*1024)/1000000.0
    //bsumdup := float64((fusage.Blocks/2)*1024)/1000000.0

    fmt.Printf("%v: %v files (%vd %vs %vh) %.6f MB (%.2f MBK)\n",
        dir,
        fusage.Files,
        fusage.Dirs,
        fusage.SymLink,
        fusage.HLinks,
        float64(fusage.Size)/1000000.0,bsume);
}
const (
    S_IFMT    = 0170000
    S_IFCHR   = 0020000
    S_IFDIR   = 0040000
    S_IFREG   = 0100000
    S_IFLNK   = 0120000
    S_IFSOCK  = 0140000
```

```
)
func cumFinfo(fsum *fileSum, path string, staterr error, fstat syscall.Stat_t, argv[]string,verb bool)(*fileSum){
    now := time.Now()
    if time.Second <= now.Sub(fsum.nextRepo) {
        if !fsum.nextRepo.IsZero(){
            tstmp := now.Format(time.Stamp)
            showFusage(tstmp,fsum)
        }
        fsum.nextRepo = now.Add(time.Second)
    }
    if staterr != nil {
        fsum.Err += 1
        return fsum
    }
    fsum.Files += 1
    if 1 < fstat.Nlink {
        // must count only once...
        // at least ignore ones in the same directory
        //if finfo.Mode().IsRegular() {
        if (fstat.Mode & S_IFMT) == S_IFREG {
            fsum.HLinks += 1
            fsum.DupBlocks += int64(fstat.Blocks)
            //fmt.Printf("---Dup HardLink %v %s\n",fstat.Nlink,path)
        }
    }
    //fsum.Size += finfo.Size()
    fsum.Size += fstat.Size
    fsum.Blocks += int64(fstat.Blocks)
    //if verb { fmt.Printf("(%8dBlk) %s",fstat.Blocks/2,path) }
    if isin("-ls",argv){
        //if verb { fmt.Printf("%4d %8d ",fstat.Blksize,fstat.Blocks) }
//      fmt.Printf("%d\t",fstat.Blocks/2)
    }
    //if finfo.IsDir()
    if (fstat.Mode & S_IFMT) == S_IFDIR {
        fsum.Dirs += 1
    }
    //if (finfo.Mode() & os.ModeSymlink) != 0
    if (fstat.Mode & S_IFMT) == S_IFLNK {
        //if verb { fmt.Printf("symlink(%v,%s)\n",fstat.Mode,finfo.Name()) }
        //{ fmt.Printf("symlink(%o,%s)\n",fstat.Mode,finfo.Name()) }
        fsum.SymLink += 1
    }
    return fsum
}
func xxFindEntv(gshCtx GshContext,depth int,total *fileSum,dir string, dstat syscall.Stat_t, ei int, entv []string,npatv[]string,argv[]string)(GshContext,*fileSum){
    nols := isin("-grep",argv)
    // sort entv
    /*
    if isin("-t",argv){
        sort.Slice(filev, func(i,j int) bool {
            return 0 < filev[i].ModTime().Sub(filev[j].ModTime())
        })
    }
    */
        /*
        if isin("-u",argv){
            sort.Slice(filev, func(i,j int) bool {
                return 0 < filev[i].AccTime().Sub(filev[j].AccTime())
            })
        }
        if isin("-U",argv){
            sort.Slice(filev, func(i,j int) bool {
                return 0 < filev[i].CreatTime().Sub(filev[j].CreatTime())
            })
        }
        */
    /*
    if isin("-S",argv){
        sort.Slice(filev, func(i,j int) bool {
            return filev[j].Size() < filev[i].Size()
        })
    }
    */
    for _,filename := range entv {
        for _,npat := range npatv {
            match := true
            if npat == "*" {
                match = true
            }else{
                match, _ = filepath.Match(npat,filename)
            }
            path := dir + DIRSEP + filename
            if !match {
                continue
            }
            var fstat syscall.Stat_t
            staterr := syscall.Lstat(path,&fstat)
            if staterr != nil {
                if !isin("-w",argv){fmt.Printf("ufind: %v\n",staterr) }
                continue;
            }
            if isin("-du",argv) && (fstat.Mode & S_IFMT) == S_IFDIR {
                // should not show size of directory in "-du" mode ...
            }else
            if !nols && !isin("-s",argv) && (!isin("-du",argv) || isin("-a",argv)) {
                if isin("-du",argv) {
                    fmt.Printf("%d\t",fstat.Blocks/2)
                }
                showFileInfo(path,argv)
            }
            if true { // && isin("-du",argv)
                total = cumFinfo(total,path,staterr,fstat,argv,false)
            }
            /*
            if isin("-wc",argv) {
            }
            */
            x := isinX("-grep",argv); // -grep will be convenient like -ls
            if 0 <= x && x+1 <= len(argv) { // -grep will be convenient like -ls
                if IsRegFile(path){
                    found := xGrep(gshCtx,path,argv[x+1:])
                    if 0 < found {
                        foundv := gshCtx.CmdCurrent.FoundFile
                        if len(foundv) < 10 {
                            gshCtx.CmdCurrent.FoundFile =
                            append(gshCtx.CmdCurrent.FoundFile,path)
                        }
                    }
                }
            }
            if !isin("-r0",argv) { // -d 0 in du, -depth n in find
                //total.Depth += 1
                if (fstat.Mode & S_IFMT) == S_IFLNK {
                    continue
                }
                if dstat.Rdev != fstat.Rdev {
                    fmt.Printf("--I-- don't follow differnet device %v(%v) %v(%v)\n",
                        dir,dstat.Rdev,path,fstat.Rdev)
                }
                if (fstat.Mode & S_IFMT) == S_IFDIR {
                    gshCtx,total = xxFind(gshCtx,depth+1,total,path,npatv,argv)
                }
            }
        }
```

```
676     }
677     return gshCtx,total
678 }
679 func xxFind(gshCtx GshContext,depth int,total *fileSum,dir string,npatv[]string,argv[]string)(GshContext,*fileSum){
680     nols := isin("-grep",argv)
681     dirfile,oerr := os.OpenFile(dir,os.O_RDONLY,0)
682     if oerr == nil {
683         //fmt.Printf("--I-- %v(%v)[%d]\n",dir,dirfile,dirfile.Fd())
684         defer dirfile.Close()
685     }else{
686     }
687
688     prev := *total
689     var dstat syscall.Stat_t
690     staterr := syscall.Lstat(dir,&dstat) // should be flstat
691
692     if staterr != nil {
693         if !isin("-w",argv){ fmt.Printf("ufind: %v\n",staterr) }
694         return gshCtx,total
695     }
696         //filev,err := ioutil.ReadDir(dir)
697         //_,err := ioutil.ReadDir(dir) // ReadDir() heavy and bad for huge directory
698         /*
699         if err != nil {
700             if !isin("-w",argv){ fmt.Printf("ufind: %v\n",err) }
701             return total
702         }
703         */
704     if depth == 0 {
705         total = cumFinfo(total,dir,staterr,dstat,argv,true)
706         if !nols && !isin("-s",argv) && (!isin("-du",argv) || isin("-a",argv)) {
707             showFileInfo(dir,argv)
708         }
709     }
710     // it it is not a directory, just scan it and finish
711
712     for ei := 0; ; ei++ {
713         entv,rderr := dirfile.Readdirnames(8*1024)
714         if len(entv) == 0 || rderr != nil {
715             //if rderr != nil { fmt.Printf("[%d] len=%d (%v)\n",ei,len(entv),rderr) }
716             break
717         }
718         if 0 < ei {
719             fmt.Printf("--I-- xxFind[%d] %d large-dir: %s\n",ei,len(entv),dir)
720         }
721         gshCtx,total = xxFindEntv(gshCtx,depth,total,dir,dstat,ei,entv,npatv,argv)
722     }
723     if isin("-du",argv) {
724         // if in "du" mode
725         fmt.Printf("%d\t%s\n",(total.Blocks-prev.Blocks)/2,dir)
726     }
727     return gshCtx,total
728 }
729
730 // {ufind|fu|ls} [Files] [// Names] [-- Expressions]
731 //  Files is "." by default
732 //  Names is "*" by default
733 //  Expressions is "-print" by default for "ufind", or -du for "fu" command
734 func xFind(gshCtx GshContext,argv[]string)(GshContext){
735     if 0 < len(argv) && strBegins(argv[0],"?"){
736         showFound(gshCtx,argv)
737         return gshCtx
738     }
739     var total = fileSum{}
740     npats := []string{}
741     for _,v := range argv {
742         if 0 < len(v) && v[0] != '-' {
743             npats = append(npats,v)
744         }
745         if v == "//" { break }
746         if v == "--" { break }
747         if v == "-grep" { break }
748         if v == "-ls" { break }
749     }
750     if len(npats) == 0 {
751         npats = []string{"*"}
752     }
753     cwd := "."
754     // if to be fullpath ::: cwd, _ := os.Getwd()
755     if len(npats) == 0 { npats = []string{"*"} }
756     gshCtx,fusage := xxFind(gshCtx,0,&total,cwd,npats,argv)
757     if !isin("-grep",argv) {
758         showFusage("total",fusage)
759     }
760     return gshCtx
761 }
762
763 func showFiles(files[]string){
764     sp := ""
765     for i,file := range files {
766         if 0 < i { sp = " " } else { sp = "" }
767         fmt.Printf(sp+"%s",escapeWhiteSP(file))
768     }
769 }
770 func showFound(gshCtx GshContext, argv[]string){
771     for i,v := range gshCtx.CommandHistory {
772         if 0 < len(v.FoundFile) {
773             fmt.Printf("!%d (%d) ",i,len(v.FoundFile))
774             if isin("-ls",argv){
775                 fmt.Printf("\n")
776                 for _,file := range v.FoundFile {
777                     fmt.Printf("") //sub number?
778                     showFileInfo(file,argv)
779                 }
780             }else{
781                 showFiles(v.FoundFile)
782                 fmt.Printf("\n")
783             }
784         }
785     }
786 }
787
788 func showMatchFile(filev []os.FileInfo, npat,dir string, argv[]string)(string,bool){
789     fname := ""
790     found := false
791     for _,v := range filev {
792         match, _ := filepath.Match(npat,(v.Name()))
793         if match {
794             fname = v.Name()
795             found = true
796             //fmt.Printf("[%d] %s\n",i,v.Name())
797             showIfExecutable(fname,dir,argv)
798         }
799     }
800     return fname,found
801 }
802 func showIfExecutable(name,dir string,argv[]string)(ffullpath string,ffound bool){
803     var fullpath string
804     if strBegins(name,DIRSEP){
805         fullpath = name
806     }else{
807         fullpath = dir + DIRSEP + name
808     }
809     fi, err := os.Stat(fullpath)
810     if err != nil {
```

```
811         fullpath = dir + DIRSEP + name + ".go"
812         fi, err = os.Stat(fullpath)
813     }
814     if err == nil {
815         fm := fi.Mode()
816         if fm.IsRegular() {
817           // R_OK=4, W_OK=2, X_OK=1, F_OK=0
818           if syscall.Access(fullpath,5) == nil {
819             ffullpath = fullpath
820             ffound = true
821             if ! isin("-s", argv) {
822                 showFileInfo(fullpath,argv)
823             }
824           }
825         }
826     }
827     return ffullpath, ffound
828 }
829 func which(list string, argv []string) (fullpathv []string, itis bool){
830     if len(argv) <= 1 {
831         fmt.Printf("Usage: which comand [-s] [-a] [-ls]\n")
832         return []string{""}, false
833     }
834     path := argv[1]
835     if strBegins(path,"/") {
836         // should check if excecutable?
837         _,exOK := showIfExecutable(path,"/",argv)
838         fmt.Printf("--D-- %v exOK=%v\n",path,exOK)
839         return []string{path},exOK
840     }
841     pathenv, efound := os.LookupEnv(list)
842     if ! efound {
843         fmt.Printf("--E-- which: no \"%s\" environment\n",list)
844         return []string{""}, false
845     }
846     showall := isin("-a",argv) || 0 <= strings.Index(path,"*")
847     dirv := strings.Split(pathenv,PATHSEP)
848     ffound := false
849     ffullpath := path
850     for _, dir := range dirv {
851         if 0 <= strings.Index(path,"*") { // by wild-card
852             list,_ := ioutil.ReadDir(dir)
853             ffullpath, ffound = showMatchFile(list,path,dir,argv)
854         }else{
855             ffullpath, ffound = showIfExecutable(path,dir,argv)
856         }
857         //if ffound && !isin("-a", argv) {
858         if ffound && !showall {
859             break;
860         }
861     }
862     return []string{ffullpath}, ffound
863 }
864
865 func stripLeadingWSParg(argv[]string)([]string){
866     for ; 0 < len(argv); {
867         if len(argv[0]) == 0 {
868             argv = argv[1:]
869         }else{
870             break
871         }
872     }
873     return argv
874 }
875 func xEval(argv []string, nlend bool){
876     argv = stripLeadingWSParg(argv)
877     if len(argv) == 0 {
878         fmt.Printf("eval [%%format] [Go-expression]\n")
879         return
880     }
881     pfmt := "%v"
882     if argv[0][0] == '%' {
883         pfmt = argv[0]
884         argv = argv[1:]
885     }
886     if len(argv) == 0 {
887         return
888     }
889     gocode := strings.Join(argv," ");
890     //fmt.Printf("eval [%v] [%v]\n",pfmt,gocode)
891     fset := token.NewFileSet()
892     rval, _ := types.Eval(fset,nil,token.NoPos,gocode)
893     fmt.Printf(pfmt,rval.Value)
894     if nlend { fmt.Printf("\n") }
895 }
896
897 func getval(name string) (found bool, val int) {
898     /* should expand the name here */
899     if name == "gsh.pid" {
900         return true, os.Getpid()
901     }else
902     if name == "gsh.ppid" {
903         return true, os.Getppid()
904     }
905     return false, 0
906 }
907
908 func echo(argv []string, nlend bool){
909     for ai := 1; ai < len(argv); ai++ {
910         if 1 < ai {
911             fmt.Printf(" ");
912         }
913         arg := argv[ai]
914         found, val := getval(arg)
915         if found {
916             fmt.Printf("%d",val)
917         }else{
918             fmt.Printf("%s",arg)
919         }
920     }
921     if nlend {
922         fmt.Printf("\n");
923     }
924 }
925
926 func resfile() string {
927     return "gsh.tmp"
928 }
929 //var resF *File
930 func resmap() {
931     //_ , err := os.OpenFile(resfile(), os.O_RDWR|os.O_CREATE, os.ModeAppend)
932     // https://developpaper.com/solution-to-golang-bad-file-descriptor-problem/
933     _ , err := os.OpenFile(resfile(), os.O_RDWR|os.O_CREATE, 0600)
934     if err != nil {
935         fmt.Printf("refF could not open: %s\n",err)
936     }else{
937         fmt.Printf("refF opened\n")
938     }
939 }
940
941 // <a name=ex-commands>External commands</a>
942 func excommand(gshCtx GshContext, exec bool, argv []string) (GshContext, bool) {
943     if gshCtx.CmdTrace { fmt.Printf("--I-- excommand[%v](%v)\n",exec,argv) }
944
945     gshPA := gshCtx.gshPA
```

```
946     fullpathv, itis := which("PATH",[]string{"which",argv[0],"-s"})
947     if itis == false {
948         return gshCtx, true
949     }
950     fullpath := fullpathv[0]
951     argv = unescapeWhiteSPV(argv)
952     if 0 < strings.Index(fullpath,".go") {
953         nargv := argv // []string{}
954         gofullpathv, itis := which("PATH",[]string{"which","go","-s"})
955         if itis == false {
956             fmt.Printf("--F-- Go not found\n")
957             return gshCtx, true
958         }
959         gofullpath := gofullpathv[0]
960         nargv = []string{ gofullpath, "run", fullpath }
961         fmt.Printf("--I-- %s {%s %s %s}\n",gofullpath,
962             nargv[0],nargv[1],nargv[2])
963         if exec {
964             syscall.Exec(gofullpath,nargv,os.Environ())
965         }else{
966             pid, _ := syscall.ForkExec(gofullpath,nargv,&gshPA)
967             if gshCtx.BackGround {
968                 fmt.Printf("--I-- in Background [%d]\n",pid)
969                 gshCtx.BackGroundJobs = append(gshCtx.BackGroundJobs,pid)
970             }else{
971                 rusage := syscall.Rusage {}
972                 syscall.Wait4(pid,nil,0,&rusage)
973                 gshCtx.LastRusage = rusage
974                 gshCtx.CmdCurrent.Rusagev[1] = rusage
975             }
976         }
977     }else{
978         if exec {
979             syscall.Exec(fullpath,argv,os.Environ())
980         }else{
981             pid, _ := syscall.ForkExec(fullpath,argv,&gshPA)
982             //fmt.Printf("[%d]\n",pid); // '&' to be background
983             if gshCtx.BackGround {
984                 fmt.Printf("--I-- in Background [%d]\n",pid)
985                 gshCtx.BackGroundJobs = append(gshCtx.BackGroundJobs,pid)
986             }else{
987                 rusage := syscall.Rusage {}
988                 syscall.Wait4(pid,nil,0,&rusage);
989                 gshCtx.LastRusage = rusage
990                 gshCtx.CmdCurrent.Rusagev[1] = rusage
991             }
992         }
993     }
994     return gshCtx, false
995 }
996
997 // <a name=builtin>Builtin Commands</a>
998 func sleep(gshCtx GshContext, argv []string) {
999     if len(argv) < 2 {
1000         fmt.Printf("Sleep 100ms, 100us, 100ns, ...\n")
1001         return
1002     }
1003     duration := argv[1];
1004     d, err := time.ParseDuration(duration)
1005     if err != nil {
1006         d, err = time.ParseDuration(duration+"s")
1007         if err != nil {
1008             fmt.Printf("duration ? %s (%s)\n",duration,err)
1009             return
1010         }
1011     }
1012     //fmt.Printf("Sleep %v\n",duration)
1013     time.Sleep(d)
1014     if 0 < len(argv[2:]) {
1015         gshellv(gshCtx, argv[2:])
1016     }
1017 }
1018 func repeat(gshCtx GshContext, argv []string) {
1019     if len(argv) < 2 {
1020         return
1021     }
1022     start0 := time.Now()
1023     for ri,_ := strconv.Atoi(argv[1]); 0 < ri; ri-- {
1024         if 0 < len(argv[2:]) {
1025             //start := time.Now()
1026             gshellv(gshCtx, argv[2:])
1027             end := time.Now()
1028             elps := end.Sub(start0);
1029             if( 1000000000 < elps ){
1030                 fmt.Printf("(repeat#%d %v)\n",ri,elps);
1031             }
1032         }
1033     }
1034 }
1035
1036 func gen(gshCtx GshContext, argv []string) {
1037     gshPA := gshCtx.gshPA
1038     if len(argv) < 2 {
1039         fmt.Printf("Usage: %s N\n",argv[0])
1040         return
1041     }
1042     // should br repeated by "repeat" command
1043     count, _ := strconv.Atoi(argv[1])
1044     fd := gshPA.Files[1] // Stdout
1045     file := os.NewFile(fd,"internalStdOut")
1046     fmt.Printf("--I-- Gen. Count=%d to [%d]\n",count,file.Fd())
1047     //buf := []byte{}
1048     outdata := "0123 5678 0123 5678 0123 5678 0123 5678\r"
1049     for gi := 0; gi < count; gi++ {
1050         file.WriteString(outdata)
1051     }
1052     //file.WriteString("\n")
1053     fmt.Printf("\n(%d B)\n",count*len(outdata));
1054     //file.Close()
1055 }
1056
1057 // <a name=network>network</a>
1058 // -s, -si, -so // bi-directional, source, sync (maybe socket)
1059 func sconnect(gshCtx GshContext, inTCP bool, argv []string) {
1060     gshPA := gshCtx.gshPA
1061     if len(argv) < 2 {
1062         fmt.Printf("Usage: -s [host]:[port[.udp]]\n")
1063         return
1064     }
1065     remote := argv[1]
1066     if remote == ":" { remote = "0.0.0.0:9999" }
1067
1068     if inTCP { // TCP
1069         dport, err := net.ResolveTCPAddr("tcp",remote);
1070         if err != nil {
1071             fmt.Printf("Address error: %s (%s)\n",remote,err)
1072             return
1073         }
1074         conn, err := net.DialTCP("tcp",nil,dport)
1075         if err != nil {
1076             fmt.Printf("Connection error: %s (%s)\n",remote,err)
1077             return
1078         }
1079         file, _ := conn.File();
1080         fd := file.Fd()
```

```
        fmt.Printf("Socket: connected to %s, socket[%d]\n",remote,fd)

        savfd := gshPA.Files[1]
        gshPA.Files[1] = fd;
        gshellv(gshCtx, argv[2:])
        gshPA.Files[1] = savfd
        file.Close()
        conn.Close()
    }else{
        //dport, err := net.ResolveUDPAddr("udp4",remote);
        dport, err := net.ResolveUDPAddr("udp",remote);
        if err != nil {
            fmt.Printf("Address error: %s (%s)\n",remote,err)
            return
        }
        //conn, err := net.DialUDP("udp4",nil,dport)
        conn, err := net.DialUDP("udp",nil,dport)
        if err != nil {
            fmt.Printf("Connection error: %s (%s)\n",remote,err)
            return
        }
        file, _ := conn.File();
        fd := file.Fd()

        ar := conn.RemoteAddr()
        //al := conn.LocalAddr()
        fmt.Printf("Socket: connected to %s [%s], socket[%d]\n",
            remote,ar.String(),fd)

        savfd := gshPA.Files[1]
        gshPA.Files[1] = fd;
        gshellv(gshCtx, argv[2:])
        gshPA.Files[1] = savfd
        file.Close()
        conn.Close()
    }
}
func saccept(gshCtx GshContext, inTCP bool, argv []string) {
    gshPA := gshCtx.gshPA
    if len(argv) < 2 {
        fmt.Printf("Usage: -ac [host]:[port[.udp]]\n")
        return
    }
    local := argv[1]
    if local == ":" { local = "0.0.0.0:9999" }
    if inTCP { // TCP
        port, err := net.ResolveTCPAddr("tcp",local);
        if err != nil {
            fmt.Printf("Address error: %s (%s)\n",local,err)
            return
        }
        //fmt.Printf("Listen at %s...\n",local);
        sconn, err := net.ListenTCP("tcp", port)
        if err != nil {
            fmt.Printf("Listen error: %s (%s)\n",local,err)
            return
        }
        //fmt.Printf("Accepting at %s...\n",local);
        aconn, err := sconn.AcceptTCP()
        if err != nil {
            fmt.Printf("Accept error: %s (%s)\n",local,err)
            return
        }
        file, _ := aconn.File()
        fd := file.Fd()
        fmt.Printf("Accepted TCP at %s [%d]\n",local,fd)

        savfd := gshPA.Files[0]
        gshPA.Files[0] = fd;
        gshellv(gshCtx, argv[2:])
        gshPA.Files[0] = savfd

        sconn.Close();
        aconn.Close();
        file.Close();
    }else{
        //port, err := net.ResolveUDPAddr("udp4",local);
        port, err := net.ResolveUDPAddr("udp",local);
        if err != nil {
            fmt.Printf("Address error: %s (%s)\n",local,err)
            return
        }
        fmt.Printf("Listen UDP at %s...\n",local);
        //uconn, err := net.ListenUDP("udp4", port)
        uconn, err := net.ListenUDP("udp", port)
        if err != nil {
            fmt.Printf("Listen error: %s (%s)\n",local,err)
            return
        }
        file, _ := uconn.File()
        fd := file.Fd()
        ar := uconn.RemoteAddr()
        remote := ""
        if ar != nil { remote = ar.String() }
        if remote == "" { remote = "?" }

        // not yet received
        //fmt.Printf("Accepted at %s [%d] <- %s\n",local,fd,"")

        savfd := gshPA.Files[0]
        gshPA.Files[0] = fd;
        savenv := gshPA.Env
        gshPA.Env = append(savenv, "REMOTE_HOST="+remote)
        gshellv(gshCtx, argv[2:])
        gshPA.Env = savenv
        gshPA.Files[0] = savfd

        uconn.Close();
        file.Close();
    }
}

// empty line command
func xPwd(gshCtx GshContext, argv[]string){
    // execute context command, pwd + date
    // context notation, representation scheme, to be resumed at re-login
    cwd, _ := os.Getwd()
    switch {
    case isin("-a",argv):
        xChdirHistory(gshCtx,argv)
    case isin("-ls",argv):
        showFileInfo(cwd,argv)
    default:
        fmt.Printf("%s\n",cwd)
    case isin("-v",argv): // obsolete emtpy command
        t := time.Now()
        date := t.Format(time.UnixDate)
        exe, _ := os.Executable()
        host, _ := os.Hostname()
        fmt.Printf("{PWD=\"%s\"",cwd)
        fmt.Printf(" HOST=\"%s\"",host)
        fmt.Printf(" DATE=\"%s\"",date)
        fmt.Printf(" TIME=\"%s\"",t.String())
        fmt.Printf(" PID=\"%d\"",os.Getpid())
        fmt.Printf(" EXE=\"%s\"",exe)
```

```
1216        fmt.Printf("}\n")
1217    }
1218 }
1219
1220 // <a name=history>History</a>
1221 // these should be browsed and edited by HTTP browser
1222 // show the time of command with -t and direcotry with -ls
1223 // openfile-history, sort by -a -m -c
1224 // sort by elapsed time by -t -s
1225 // search by "more" like interface
1226 // edit history
1227 // sort history, and wc or uniq
1228 // CPU and other resource consumptions
1229 // limit showing range (by time or so)
1230 // export / import history
1231 func xHistory(gshCtx GshContext, argv []string) (rgshCtx GshContext) {
1232     for i, v := range gshCtx.CommandHistory {
1233         // exclude commands not to be listed by default
1234         // internal commands may be suppressed by default
1235         if v.CmdLine == "" && !isin("-a",argv) {
1236             continue;
1237         }
1238         if !isin("-n",argv){ // like "fc"
1239             fmt.Printf("!%-3d ",i)
1240         }
1241         if isin("-v",argv){
1242             fmt.Println(v) // should be with it date
1243         }else{
1244             if isin("-l",argv) || isin("-l0",argv) {
1245                 elps := v.EndAt.Sub(v.StartAt);
1246                 start := v.StartAt.Format(time.Stamp)
1247                 fmt.Printf("%s %11v/t ",start,elps)
1248             }
1249             if isin("-l",argv) && !isin("-l0",argv){
1250                 fmt.Printf("%v",Rusagef("%t %u %s",argv,v.Rusagev))
1251             }
1252             if isin("-ls",argv){
1253                 fmt.Printf("@%s ",v.WorkDir)
1254                 // show the FileInfo of the output command??
1255             }
1256             fmt.Printf("%s",v.CmdLine)
1257             fmt.Printf("\n")
1258         }
1259     }
1260     return gshCtx
1261 }
1262 // !n - history index
1263 func searchHistory(gshCtx GshContext, gline string) (string, bool, bool){
1264     if gline[0] == '!' {
1265         hix, err := strconv.Atoi(gline[1:])
1266         if err != nil {
1267             fmt.Printf("--E-- (%s : range)\n",hix)
1268             return "", false, true
1269         }
1270         if hix < 0 || len(gshCtx.CommandHistory) <= hix {
1271             fmt.Printf("--E-- (%d : out of range)\n",hix)
1272             return "", false, true
1273         }
1274         return gshCtx.CommandHistory[hix].CmdLine, false, false
1275     }
1276     // search
1277     //for i, v := range gshCtx.CommandHistory {
1278     //}
1279     return gline, false, false
1280 }
1281
1282 // temporary adding to PATH environment
1283 // cd name -lib for LD_LIBRARY_PATH
1284 // chdir with directory history (date + full-path)
1285 // -s for sort option (by visit date or so)
1286 func xChdirHistory(gshCtx GshContext, argv []string){
1287     for i, v := range gshCtx.ChdirHistory {
1288         fmt.Printf("!%d ",i)
1289         fmt.Printf("%v ",v.MovedAt.Format(time.Stamp))
1290         showFileInfo(v.Dir,argv)
1291     }
1292 }
1293 func xChdir(gshCtx GshContext, argv []string) (rgshCtx GshContext) {
1294     cdhist := gshCtx.ChdirHistory
1295     if isin("?",argv ) || isin("-t",argv) {
1296         xChdirHistory(gshCtx,argv)
1297         return gshCtx
1298     }
1299     pwd, _ := os.Getwd()
1300     dir := ""
1301     if len(argv) <= 1 {
1302         dir = toFullpath("~")
1303     }else{
1304         dir = argv[1]
1305     }
1306     if strBegins(dir,"!") {
1307         if dir == "!0" {
1308             dir = gshCtx.StartDir
1309         }else
1310         if dir == "!!" {
1311             index := len(cdhist) - 1
1312             if 0 < index { index -= 1 }
1313             dir = cdhist[index].Dir
1314         }else{
1315             index, err := strconv.Atoi(dir[1:])
1316             if err != nil {
1317                 fmt.Printf("--E-- xChdir(%v)\n",err)
1318                 dir = "?"
1319             }else
1320             if len(gshCtx.ChdirHistory) <= index {
1321                 fmt.Printf("--E-- xChdir(history range error)\n")
1322                 dir = "?"
1323             }else{
1324                 dir = cdhist[index].Dir
1325             }
1326         }
1327     }
1328     if dir != "?" {
1329         err := os.Chdir(dir)
1330         if err != nil {
1331             fmt.Printf("--E-- xChdir(%s)(%v)\n",argv[1],err)
1332         }else{
1333             cwd, _ := os.Getwd()
1334             if cwd != pwd {
1335                 hist1 := GChdirHistory { }
1336                 hist1.Dir = cwd
1337                 hist1.MovedAt = time.Now()
1338                 gshCtx.ChdirHistory = append(cdhist,hist1)
1339             }
1340         }
1341     }
1342     if isin("-ls",argv){
1343         cwd, _ := os.Getwd()
1344         showFileInfo(cwd,argv);
1345     }
1346     return gshCtx
1347 }
1348 func TimeValSub(tv1 *syscall.Timeval, tv2 *syscall.Timeval){
1349     *tv1 = syscall.NsecToTimeval(tv1.Nano() - tv2.Nano())
1350 }
```

```
func RusageSubv(ru1, ru2 [2]syscall.Rusage)([2]syscall.Rusage){
    TimeValSub(&ru1[0].Utime,&ru2[0].Utime)
    TimeValSub(&ru1[0].Stime,&ru2[0].Stime)
    TimeValSub(&ru1[1].Utime,&ru2[1].Utime)
    TimeValSub(&ru1[1].Stime,&ru2[1].Stime)
    return ru1
}
func TimeValAdd(tv1 syscall.Timeval, tv2 syscall.Timeval)(syscall.Timeval){
    tvs := syscall.NsecToTimeval(tv1.Nano() + tv2.Nano())
    return tvs
}
/*
func RusageAddv(ru1, ru2 [2]syscall.Rusage)([2]syscall.Rusage){
    TimeValAdd(ru1[0].Utime,ru2[0].Utime)
    TimeValAdd(ru1[0].Stime,ru2[0].Stime)
    TimeValAdd(ru1[1].Utime,ru2[1].Utime)
    TimeValAdd(ru1[1].Stime,ru2[1].Stime)
    return ru1
}
*/

// <a name=rusage>Resource Usage</a>
func Rusagef(fmtspec string, argv []string, ru [2]syscall.Rusage)(string){
    ut := TimeValAdd(ru[0].Utime,ru[1].Utime)
    st := TimeValAdd(ru[0].Stime,ru[1].Stime)
    fmt.Printf("%d.%06ds/u ",ut.Sec,ut.Usec) //ru[1].Utime.Sec,ru[1].Utime.Usec)
    fmt.Printf("%d.%06ds/s ",st.Sec,st.Usec) //ru[1].Stime.Sec,ru[1].Stime.Usec)
    return ""
}
func Getrusagev()([2]syscall.Rusage){
    var ruv = [2]syscall.Rusage{}
    syscall.Getrusage(syscall.RUSAGE_SELF,&ruv[0])
    syscall.Getrusage(syscall.RUSAGE_CHILDREN,&ruv[1])
    return ruv
}
func showRusage(what string,argv []string, ru *syscall.Rusage){
    fmt.Printf("%s: ",what);
    fmt.Printf("Usr=%d.%06ds",ru.Utime.Sec,ru.Utime.Usec)
    fmt.Printf(" Sys=%d.%06ds",ru.Stime.Sec,ru.Stime.Usec)
    fmt.Printf(" Rss=%vB",ru.Maxrss)
    if isin("-l",argv) {
        fmt.Printf(" MinFlt=%v",ru.Minflt)
        fmt.Printf(" MajFlt=%v",ru.Majflt)
        fmt.Printf(" IxRSS=%vB",ru.Ixrss)
        fmt.Printf(" IdRSS=%vB",ru.Idrss)
        fmt.Printf(" Nswap=%vB",ru.Nswap)
    fmt.Printf(" Read=%v",ru.Inblock)
    fmt.Printf(" Write=%v",ru.Oublock)
    }
    fmt.Printf(" Snd=%v",ru.Msgsnd)
    fmt.Printf(" Rcv=%v",ru.Msgrcv)
    //if isin("-l",argv) {
        fmt.Printf(" Sig=%v",ru.Nsignals)
    //}
    fmt.Printf("\n");
}
func xTime(gshCtx GshContext, argv[]string)(GshContext,bool){
    if 2 <= len(argv){
        gshCtx.LastRusage = syscall.Rusage{}
        rusagev1 := Getrusagev()
        xgshCtx, fin := gshellv(gshCtx,argv[1:])
        rusagev2 := Getrusagev()
        gshCtx = xgshCtx
        showRusage(argv[1],argv,&gshCtx.LastRusage)
        rusagev := RusageSubv(rusagev2,rusagev1)
        showRusage("self",argv,&rusagev[0])
        showRusage("chld",argv,&rusagev[1])
        return gshCtx, fin
    }else{
        rusage:= syscall.Rusage {}
        syscall.Getrusage(syscall.RUSAGE_SELF,&rusage)
        showRusage("self",argv, &rusage)
        syscall.Getrusage(syscall.RUSAGE_CHILDREN,&rusage)
        showRusage("chld",argv, &rusage)
        return gshCtx, false
    }
}
func xJobs(gshCtx GshContext, argv[]string){
    fmt.Printf("%d Jobs\n",len(gshCtx.BackGroundJobs))
    for ji, pid := range gshCtx.BackGroundJobs {
        //wstat := syscall.WaitStatus {0}
        rusage := syscall.Rusage {}
        //wpid, err := syscall.Wait4(pid,&wstat,syscall.WNOHANG,&rusage);
        wpid, err := syscall.Wait4(pid,nil,syscall.WNOHANG,&rusage);
        if err != nil {
            fmt.Printf("--E-- %%%d [%d] (%v)\n",ji,pid,err)
        }else{
            fmt.Printf("%%%d[%d](%d)\n",ji,pid,wpid)
            showRusage("chld",argv,&rusage)
        }
    }
}
func inBackground(gshCtx GshContext, argv[]string)(GshContext,bool){
    if gshCtx.CmdTrace { fmt.Printf("--I-- inBackground(%v)\n",argv) }
    gshCtx.BackGround = true // set background option
    xfin := false
    gshCtx, xfin = gshellv(gshCtx,argv)
    gshCtx.BackGround = false
    return gshCtx,xfin
}
// -o file without command means just opening it and refer by #N
// should be listed by "files" comnmand
func xOpen(gshCtx GshContext, argv[]string)(GshContext){
    var pv = []int{-1,-1}
    err := syscall.Pipe(pv)
    fmt.Printf("--I-- pipe()=[#%d,#%d](%v)\n",pv[0],pv[1],err)
    return gshCtx
}
func fromPipe(gshCtx GshContext, argv[]string)(GshContext){
    return gshCtx
}
func xClose(gshCtx GshContext, argv[]string)(GshContext){
    return gshCtx
}

// <a name=redirect>redirect</a>
func redirect(gshCtx GshContext, argv[]string)(GshContext,bool){
    if len(argv) < 2 {
        return gshCtx, false
    }

    cmd := argv[0]
    fname := argv[1]
    var file *os.File = nil

    fdix := 0
    mode := os.O_RDONLY

    switch {
    case cmd == "-i" || cmd == "<":
        fdix = 0
        mode = os.O_RDONLY
    case cmd == "-o" || cmd == ">":
        fdix = 1
        mode = os.O_RDWR | os.O_CREATE
```

```
    case cmd == "-a" || cmd == ">>":
        fdix = 1
        mode = os.O_RDWR | os.O_CREATE | os.O_APPEND
    }
    if fname[0] == '#' {
        fd, err := strconv.Atoi(fname[1:])
        if err != nil {
            fmt.Printf("--E-- (%v)\n",err)
            return gshCtx, false
        }
        file = os.NewFile(uintptr(fd),"MaybePipe")
    }else{
        xfile, err := os.OpenFile(argv[1], mode, 0600)
        if err != nil {
            fmt.Printf("--E-- (%s)\n",err)
            return gshCtx, false
        }
        file = xfile
    }
    gshPA := gshCtx.gshPA
    savfd := gshPA.Files[fdix]
    gshPA.Files[fdix] = file.Fd()
    fmt.Printf("--I-- Opened [%d] %s\n",file.Fd(),argv[1])
    gshCtx, _ = gshellv(gshCtx, argv[2:])
    gshPA.Files[fdix] = savfd

    return gshCtx, false
}

//fmt.Fprintf(res, "GShell Status: %q", html.EscapeString(req.URL.Path))
func httpHandler(res http.ResponseWriter, req *http.Request){
    path := req.URL.Path
    fmt.Printf("--I-- Got HTTP Request(%s)\n",path)
    {
        gshCtx, _ :=  setupGshContext()
        fmt.Printf("--I-- %s\n",path[1:])
        gshCtx, _ = tgshelll(gshCtx,path[1:])
    }
    fmt.Fprintf(res, "Hello(^-^)/\n%s\n",path)
}
func httpServer(gshCtx GshContext, argv []string){
    http.HandleFunc("/", httpHandler)
    accport := "localhost:9999"
    fmt.Printf("--I-- HTTP Server Start at [%s]\n",accport)
    http.ListenAndServe(accport,nil)
}
func xGo(gshCtx GshContext, argv[]string){
    go gshellv(gshCtx,argv[1:]);
}
func xPs(gshCtx GshContext, argv[]string)(GshContext){
    return gshCtx
}

// <a name=plugin>Plugin</a>
// plugin [-ls [names]] to list plugins
// Reference: <a href=https://golang.org/src/plugin/>plugin</a> source code
func whichPlugin(gshCtx GshContext,name string,argv[]string)(pi *PluginInfo){
    pi = nil
    for _,p := range gshCtx.PluginFuncs {
        if p.Name == name && pi == nil {
            pi = &p
        }
        if !isin("-s",argv){
            //fmt.Printf("%v %v ",i,p)
            if isin("-ls",argv){
                showFileInfo(p.Path,argv)
            }else{
                fmt.Printf("%s\n",p.Name)
            }
        }
    }
    return pi
}
func xPlugin(gshCtx GshContext, argv[]string)(GshContext,error){
    if len(argv) == 0 || argv[0] == "-ls" {
        whichPlugin(gshCtx,"",argv)
        return gshCtx, nil
    }
    name := argv[0]
    Pin := whichPlugin(gshCtx,name,[]string{"-s"})
    if Pin != nil {
        os.Args = argv // should be recovered?
        Pin.Addr.(func())()
        return gshCtx,nil
    }
    sofile := toFullpath(argv[0] + ".so") // or find it by which($PATH)

    p, err := plugin.Open(sofile)
    if err != nil {
        fmt.Printf("--E-- plugin.Open(%s)(%v)\n",sofile,err)
        return gshCtx, err
    }
    fname := "Main"
    f, err := p.Lookup(fname)
    if( err != nil ){
        fmt.Printf("--E-- plugin.Lookup(%s)(%v)\n",fname,err)
        return gshCtx, err
    }
    pin := PluginInfo {p,f,name,sofile}
    gshCtx.PluginFuncs = append(gshCtx.PluginFuncs,pin)
    fmt.Printf("--I-- added (%d)\n",len(gshCtx.PluginFuncs))

    //fmt.Printf("--I-- first call(%s:%s)%v\n",sofile,fname,argv)
    os.Args = argv
    f.(func())()
    return gshCtx, err
}
func Args(gshCtx *GshContext, argv[]string){
    for i,v := range os.Args {
        fmt.Printf("[%v] %v\n",i,v)
    }
}
func Version(gshCtx *GshContext, argv[]string){
    if isin("-l",argv) {
        fmt.Printf("%v/%v (%v)",NAME,VERSION,DATE);
    }else{
        fmt.Printf("%v",VERSION);
    }
    if !isin("-n",argv) {
        fmt.Printf("\n")
    }
}

// <a name=interpreter>Command Interpreter</a>
func gshellv(gshCtx GshContext, argv []string) (_ GshContext, fin bool) {
    fin = false

    if gshCtx.CmdTrace { fmt.Fprintf(os.Stderr,"--I-- gshellv((%d))\n",len(argv)) }
    if len(argv) <= 0 {
        return gshCtx, false
    }
    xargv := []string{}
    for ai := 0; ai < len(argv); ai++ {
        xargv = append(xargv,strsubst(&gshCtx,argv[ai],false))
    }
```

```
    argv = xargv
    if false {
        for ai := 0; ai < len(argv); ai++ {
            fmt.Printf("[%d] %s [%d]%T\n",
                ai,argv[ai],len(argv[ai]),argv[ai])
        }
    }
    cmd := argv[0]
    if gshCtx.CmdTrace { fmt.Fprintf(os.Stderr,"--I-- gshellv(%d)%v\n",len(argv),argv) }
    switch { // https://tour.golang.org/flowcontrol/11
    case cmd == "":
        xPwd(gshCtx,[]string{}); // emtpy command
    case cmd == "-x":
        gshCtx.CmdTrace = ! gshCtx.CmdTrace
    case cmd == "-ot":
        sconnect(gshCtx, true, argv)
    case cmd == "-ou":
        sconnect(gshCtx, false, argv)
    case cmd == "-it":
        saccept(gshCtx, true , argv)
    case cmd == "-iu":
        saccept(gshCtx, false, argv)
    case cmd == "-i" || cmd == "<" || cmd == "-o" || cmd == ">" || cmd == "-a" || cmd == ">>" || cmd == "-s" || cmd == "><":
        redirect(gshCtx, argv)
    case cmd == "|":
        gshCtx = fromPipe(gshCtx, argv)
    case cmd == "args":
        Args(&gshCtx,argv)
    case cmd == "bg" || cmd == "-bg":
        rgshCtx, rfin := inBackground(gshCtx,argv[1:])
        return rgshCtx, rfin
    case cmd == "call":
        gshCtx, _ = excommand(gshCtx, false,argv[1:])
    case cmd == "cd" || cmd == "chdir":
        gshCtx = xChdir(gshCtx,argv);
    case cmd == "close":
        gshCtx = xClose(gshCtx,argv)
    case cmd == "dec" || cmd == "decode":
        Dec(&gshCtx,argv)
    case cmd == "#define":
    case cmd == "echo":
        echo(argv,true)
    case cmd == "enc" || cmd == "encode":
        Enc(&gshCtx,argv)
    case cmd == "env":
        env(argv)
    case cmd == "eval":
        xEval(argv[1:],true)
    case cmd == "exec":
        gshCtx, _ = excommand(gshCtx, true,argv[1:])
        // should not return here
    case cmd == "exit" || cmd == "quit":
        // write Result code EXIT to 3>
        return gshCtx, true
    case cmd == "-find" || cmd == "fin" || cmd == "ufind" || cmd == "uf" || cmd == "fu":
        gshCtx = xFind(gshCtx,argv[1:])
    case cmd == "fork":
        // mainly for a server
    case cmd == "-gen":
        gen(gshCtx, argv)
    case cmd == "-go":
        xGo(gshCtx, argv)
    case cmd == "-grep":
        gshCtx = xFind(gshCtx,argv)
    case cmd == "history" || cmd == "hi": // hi should be alias
        gshCtx = xHistory(gshCtx, argv)
    case cmd == "jobs":
        xJobs(gshCtx,argv)
    case cmd == "-ls":
        gshCtx = xFind(gshCtx,argv)
    case cmd == "nop":
    case cmd == "pipe":
        gshCtx = xOpen(gshCtx,argv)
    case cmd == "plug" || cmd == "plugin" || cmd == "pin":
        gshCtx,_ = xPlugin(gshCtx,argv[1:])
    case cmd == "ps":
        xPs(gshCtx,argv)
    case cmd == "pstitle": // to be gsh.title
    case cmd == "repeat" || cmd == "rep": // repeat cond command
        repeat(gshCtx,argv)
    case cmd == "set":
        // set name ...
    case cmd == "serv":
        httpServer(gshCtx,argv)
    case cmd == "sleep":
        sleep(gshCtx,argv)
    case cmd == "lnsp":
        SplitLine(&gshCtx,argv)
    case cmd == "time":
        gshCtx, fin = xTime(gshCtx,argv)
    case cmd == "pwd":
        xPwd(gshCtx,argv);
    case cmd == "ver" || cmd == "-ver" || cmd == "version":
        Version(&gshCtx,argv)
    case cmd == "where":
        // data file or so?
    case cmd == "which":
        which("PATH",argv);
    default:
        if whichPlugin(gshCtx,cmd,[]string{"-s"}) != nil {
            gshCtx, _ = xPlugin(gshCtx,argv)
        }else{
            gshCtx, _ = excommand(gshCtx,false,argv)
        }
    }
    return gshCtx, fin
}

func gshelll(gshCtx GshContext, gline string) (gx GshContext, rfin bool) {
    argv := strings.Split(string(gline)," ")
    gshCtx, fin := gshellv(gshCtx,argv)
    return gshCtx, fin
}
func tgshelll(gshCtx GshContext, gline string) (gx GshContext, xfin bool) {
    start := time.Now()
    gshCtx, fin := gshelll(gshCtx,gline)
    end := time.Now()
    elps := end.Sub(start);
    fmt.Printf("--I-- " + time.Now().Format(time.Stamp) + "(%d.%09ds)\n",
        elps/1000000000,elps%1000000000)
    return gshCtx, fin
}
func Ttyid() (int) {
    fi, err := os.Stdin.Stat()
    if err != nil {
        return 0;
    }
    //fmt.Printf("Stdin: %v Dev=%d\n",
    //  fi.Mode(),fi.Mode()&os.ModeDevice)
    if (fi.Mode() & os.ModeDevice) != 0 {
        stat := syscall.Stat_t{};
        err := syscall.Fstat(0,&stat)
        if err != nil {
            //fmt.Printf("--I-- Stdin: (%v)\n",err)
        }else{
```

```
            //fmt.Printf("--I-- Stdin: rdev=%d %d\n",
            //  stat.Rdev&0xFF,stat.Rdev);
            //fmt.Printf("--I-- Stdin: tty%d\n",stat.Rdev&0xFF);
            return int(stat.Rdev & 0xFF)
        }
    }
    return 0
}
func ttyfile(gshCtx GshContext) string {
    //fmt.Printf("--I-- GSH_HOME=%s\n",gshCtx.GshHomeDir)
    ttyfile := gshCtx.GshHomeDir + "/" + "gsh-tty" +
        fmt.Sprintf("%02d",gshCtx.TerminalId)
        //strconv.Itoa(gshCtx.TerminalId)
    //fmt.Printf("--I-- ttyfile=%s\n",ttyfile)
    return ttyfile
}
func ttyline(gshCtx GshContext) (*os.File){
    file, err := os.OpenFile(ttyfile(gshCtx),
        os.O_RDWR|os.O_CREATE|os.O_TRUNC,0600)
    if err != nil {
        fmt.Printf("--F-- cannot open %s (%s)\n",ttyfile(gshCtx),err)
        return file;
    }
    return file
}
// <a name=getline>Command Line Editor</a>
func getline(gshCtx GshContext, hix int, skipping, with_exgetline bool, gsh_getlinev[]string, prevline string) (string) {
    if( skipping ){
        reader := bufio.NewReaderSize(os.Stdin,LINESIZE)
        line, _, _ := reader.ReadLine()
        return string(line)
    }else
    if( with_exgetline && gshCtx.GetLine != "" ){
        //var xhix int64 = int64(hix); // cast
        newenv := os.Environ()
        newenv = append(newenv, "GSH_LINENO="+strconv.FormatInt(int64(hix),10) )

        tty := ttyline(gshCtx)
        tty.WriteString(prevline)
        Pa := os.ProcAttr {
            "", // start dir
            newenv, //os.Environ(),
            []*os.File{os.Stdin,os.Stdout,os.Stderr,tty},
            nil,
        }
//fmt.Printf("--I-- getline=%s // %s\n",gsh_getlinev[0],gshCtx.GetLine)
proc, err := os.StartProcess(gsh_getlinev[0],[]string{"getline","getline"},&Pa)
        if err != nil {
            fmt.Printf("--F-- getline process error (%v)\n",err)
            // for ; ; { }
            return "exit (getline program failed)"
        }
        //stat, err := proc.Wait()
        proc.Wait()
        buff := make([]byte,LINESIZE)
        count, err := tty.Read(buff)
        //_, err = tty.Read(buff)
        //fmt.Printf("--D-- getline (%d)\n",count)
        if err != nil {
            if ! (count == 0) { // && err.String() == "EOF" ) {
                fmt.Printf("--E-- getline error (%s)\n",err)
            }
        }else{
            //fmt.Printf("--I-- getline OK \"%s\"\n",buff)
        }
        tty.Close()
        return string(buff[0:count])
    }else{
        // if isatty {
            fmt.Printf("!%d",hix)
            fmt.Print(PROMPT)
        // }
        reader := bufio.NewReaderSize(os.Stdin,LINESIZE)
        line, _, _ := reader.ReadLine()
        return string(line)
    }
}
//
// $USERHOME/.gsh/
//      gsh-rc.txt, or gsh-configure.txt
//              gsh-history.txt
//              gsh-aliases.txt // should be conditional?
//
func gshSetupHomedir(gshCtx GshContext) (GshContext, bool) {
    homedir,found := userHomeDir()
    if !found {
        fmt.Printf("--E-- You have no UserHomeDir\n")
        return gshCtx, true
    }
    gshhome := homedir + "/" + GSH_HOME
    _, err2 := os.Stat(gshhome)
    if err2 != nil {
        err3 := os.Mkdir(gshhome,0700)
        if err3 != nil {
            fmt.Printf("--E-- Could not Create %s (%s)\n",
                gshhome,err3)
            return gshCtx, true
        }
        fmt.Printf("--I-- Created %s\n",gshhome)
    }
    gshCtx.GshHomeDir = gshhome
    return gshCtx, false
}
func setupGshContext()(GshContext,bool){
    gshPA := syscall.ProcAttr {
        "", // the staring directory
        os.Environ(), // environ[]
        []uintptr{os.Stdin.Fd(),os.Stdout.Fd(),os.Stderr.Fd()},
        nil, // OS specific
    }
    cwd, _ := os.Getwd()
    gshCtx := GshContext {
        cwd, // StartDir
        "", // GetLine
        []GChdirHistory { {cwd,time.Now()} }, // ChdirHistory
        gshPA,
        []GCommandHistory{}, //something for invokation?
        GCommandHistory{}, // CmdCurrent
        false,
        []int{},
        syscall.Rusage{},
        "", // GshHomeDir
        Ttyid(),
        false,
        []PluginInfo{},
    }
    err := false
    gshCtx, err = gshSetupHomedir(gshCtx)
    return gshCtx, err
}
// <a name=main>Main loop</a>
func script(gshCtxGiven *GshContext) (_ GshContext) {
    gshCtx,err0 := setupGshContext()
    if err0 {
        return gshCtx;
```

```
    }
    //fmt.Printf("--I-- GSH_HOME=%s\n",gshCtx.GshHomeDir)
    //resmap()
    gsh_getlinev, with_exgetline :=
         which("PATH",[]string{"which","gsh-getline","-s"})
    if with_exgetline {
        gsh_getlinev[0] = toFullpath(gsh_getlinev[0])
        gshCtx.GetLine = toFullpath(gsh_getlinev[0])
    }else{
    fmt.Printf("--W-- No gsh-getline found. Using internal getline.\n");
    }

    ghist0 := gshCtx.CmdCurrent // something special, or gshrc script, or permanent history
    gshCtx.CommandHistory = append(gshCtx.CommandHistory,ghist0)

    prevline := ""
    skipping := false
    for hix := len(gshCtx.CommandHistory); ; {
        gline := getline(gshCtx,hix,skipping,with_exgetline,gsh_getlinev,prevline)
        if skipping {
            if strings.Index(gline,"fi") == 0 {
                fmt.Printf("fi\n");
                skipping = false;
            }else{
                //fmt.Printf("%s\n",gline);
            }
            continue
        }
        if strings.Index(gline,"if") == 0 {
            //fmt.Printf("--D-- if start: %s\n",gline);
            skipping = true;
            continue
        }
        gline = strsubst(&gshCtx,gline,true)
        /*
        // should be cared in substitution ?
        if 0 < len(gline) && gline[0] == '!' {
            xgline, set, err := searchHistory(gshCtx,gline)
            if err {
                continue
            }
            if set {
                // set the line in command line editor
            }
            gline = xgline
        }
        */
        ghist := gshCtx.CmdCurrent
        ghist.WorkDir,_ = os.Getwd()
        ghist.StartAt = time.Now()
        rusagev1 := Getrusagev()
        gshCtx.CmdCurrent.FoundFile = []string{}
        xgshCtx, fin := tgshelll(gshCtx,gline)
        rusagev2 := Getrusagev()
        ghist.Rusagev = RusageSubv(rusagev2,rusagev1)
        gshCtx = xgshCtx
        ghist.EndAt = time.Now()
        ghist.CmdLine = gline
        ghist.FoundFile = gshCtx.CmdCurrent.FoundFile

        /* record it but not show in list by default
        if len(gline) == 0 {
            continue
        }
        if gline == "hi" || gline == "history" { // don't record it
            continue
        }
        */
        gshCtx.CommandHistory = append(gshCtx.CommandHistory, ghist)
        if fin {
            break;
        }
        prevline = gline;
        hix++;
    }
    return gshCtx
}
func main() {
    argv := os.Args
    if 1 < len(argv) {
        if isin("version",argv){
            Version(nil,argv)
            return
        }
        comx := isinX("-c",argv)
        if 0 < comx {
            gshCtx,err := setupGshContext()
            if !err {
                gshellv(gshCtx,argv[comx+1:])
            }
            return
        }
    }
    script(nil)
    //gshCtx := script(nil)
    //gshelll(gshCtx,"time")
}
//</pre></details>
//<details id=todo><summary>Consideration</summary><pre>
// - inter gsh communication, possibly running in remote hosts -- to be remote shell
// - merged histories of multiple parallel gsh sessions
// - alias as a function
// - instant alias end environ export to the permanent > ~/.gsh/gsh-alias and gsh-environ
// - retrieval PATH of files by its type
// - gsh as an IME
// - gsh a scheduler in precise time of within a millisecond
// - all commands have its subucomand after "---" symbol
// - filename expansion by "-find" command
// - history of ext code and output of each commoand
// - "script" output for each command by pty-tee or telnet-tee
// - $BUILTIN command in PATH to show the priority
// - "?" symbol in the command (not as in arguments) shows help request
// - searching command with wild card like: which ssh-*
// - longformat prompt after long idle time (should dismiss by BS)
// - customizing by building plugin and dynamically linking it
// - generating syntactic element like "if" by macro expansion (like CPP) >> alias
// - "!" symbol should be used for negation, don't wast it just for job control
// - don't put too long output to tty, record it into GSH_HOME/session-id/comand-id.log
// - making canonical form of command at the start adding quatation or white spaces
// - name(a,b,c) ... use "(" and ")" to show both delimiter and realm
// - name? or name! might be useful
// - htar format - packing directory contents into a single html file using data scheme
//---END--- (^-^)/ITS more</pre></details>
/*
<details id=references><summary>References</summary><pre>
<p>
<a href=https://golang.org>The Go Programming Language</a>
<iframe width=100% height=300 src=https://golang.org></iframe>

<a href=https://developer.mozilla.org/ja/docs/Web>MDN web docs</a>
 <a href=https://developer.mozilla.org/ja/docs/Web/HTML/Element>HTML</a>
 CSS:
   <a href=https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Selectors>Selectors</a>
   <a href=https://developer.mozilla.org/en-US/docs/Web/CSS/background-repeat>repeat</a>
 HTTP
```

```
 JavaScript:
 ...
</p>
</pre></details>
<div id=gsh-footer>Fin.</div>
<style>
 #gsh {border-width:1;margin:0;padding:0;}
 #gsh {font-family:monospace,Courier New;color:#ddf;font-size:8px;}
 #xgsh header{height:100px;background-image:url(GShell-Logo00.png);}
 #gsh header{height:100px;}
 #gsh-footer{height:100px;background-size:50px;background-repeat:no-repeat;}
 #gsh note{color:#000;font-size:10pt;}
 #gsh h2{color:#24a;font-family:Georgia;font-size:18pt;}
 #gsh details{color:#888;background-color:#aaa;font-family:monospace;}
 #gsh summary{font-size:16pt;color:#24a;background-color:#eef;height:30px;}
 #gsh pre{font-size:11pt;color:#223;background-color:#faffff;}
 #gsh a{color:#24a;}
 #gsh a[name]{color:#24a;font-size:16pt;}
 @print {
  #gsh pre{font-size:11pt !import;}
 }
</style>
<!--
// Logo image should be drawn by JavaScript from a meta-font.
// CSS seems not follow line-splitted URL
-->
<script>
GshLogo="\
iVBORw0KGgoAAAANSUhEUgAAAQEAAAB/CAYAAADvs3f4AAAABGdBTUEAALGPC/xhBQAAACBj\
SFJNAAB6JgAAgIQAAPoAAACA6AAAdTAAAOpgAAA6mAAAF3CculE8AAAAhGVYSWZNTQAqAAAA\
CAAFARIAAwAAAAEAAQAAARoABQAAAAEAAABKARsABQAAAAEAAABSASgAAwAAAAEAAgAAh2kA\
BAAAAAEAAABaAAAAAAAAAAEgAAAABAAAASAAAAAEAA6ABAAMAAAABAAEAAKACAAQAAAABAAAB\
AaADAAQAAAABAAAAfwAAAACjqBiyAAAACXBIWXMAAAsTAAALEwEAmpwYAAACaGlUWHRYTUw6\
Y29tLmFkb2JlLnhtcAAAAAAAPHg6eGlwbWV0YSB4bWxuczp4PSJhZG9iZTpucptZXRhLyIg\
eDp4bXB0az0iWE1QIENvcmUgNS40LjAiPgogICA8cmRmOlJERiB4bWxuczpyZGY9Imh0dHA6\
Ly93d3cudzMub3JnLzE5OTkvMDIvMjItcmRmLXN5bnRheC1ucyMiPgogICAgICA8cmRmOkRl\
c2NyaXB0aW9uIHJkZjphYm91dD0iIgogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAg\
bnMuYWRvYmUuY29tL3RpZmYvMS4wLyIKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAg\
L25zLmFkb2JlLmNvbS9leGlmLzEuMC8iPgogICAgICAgICAgICAgICAgICAgICAgICAgICAg\
PC90aWZmOk9yaWVudGF0aW9uPgogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAg\
aWZmOlJlc29sdXRpb25Vbml0PgogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAg\
Q29sb3JTcGFjZT4KICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAg\
aXhlbFhEaWllbnNpb24+CiAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAg\
aWY6UGl4ZWxZRGltZW5zaW9uPgogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAg\
UkRGPgo8L3g6eGlwbWV0YT4KM5orPAAAFtpJREFUeAHtnQl0HdV5x++d9/S02AbjBWwnQChL\
AdcsMzuDJWsDAynhkBa3geSU0pQlGCy5h9CQpHXSljYBLHmBUAjLCU04ZUnKyYHDYi2WRHxI\
ChgCNg64OLaxwbvBsmS99+b2f5818rx5c2eeRrIs6f3nWJ65y3eX38z95t7v3rlPCB4kQAIk\
QAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIk\
QAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQAIkQALDlIACpuU6osU66xsv\
jZk0KT5e9iTG9BSpz2Kr09tbW6tSQlAoWXvjK0eJMSUTla3GpmX60107e7a//cTcziHImlkU\
KIGCVgKn3PZC8Qmxo6qFUBdYUp4vpDhXKTFZSlHkfh7gp6RUe5SS26RQb9lCtkvbbmta0vyO\
EItsd9z+XlfXdVyOfC9BunNwPksKGfemoZTaJ4T8rZD2qyItnmlaUvG2Nw7dJBCVQEEqgZpb\
V0yUieJblJDz0eCPiwoPjXMzFMjDPUo83N5YsbU/6VTVtV0ck/JeIeVF/ZHTcaGTfiPS6bqm\
JZW/C5KdeeN/Fo0rO216XMYnQVOVSlt2p2Vyulgt3hminklQ8QYUFqVuUWQGVMgRIlxQSqDi\
9pWnJuKxerzZr0fjLx2se4T0kugp/HhD8rN/+WDZFQeC0q2sbzklJor+A/n/RVC8sDCdZlKl\
z25rnLPWL27Nre2nyYTogJKZnBOu1GsrGsr7rXxy0jlCHlHqFkXmCFVvyLPN6XoOeQmGIMMv\
3fzSsaWlZQ9KKa9Cduj5D26mSA/DB/ndE+LjlAdCfN+UenVd+zfQ3X/kYHxTrPz8dRoJJf8E\
sX2VAEr0574KIJO8nJhfLsM0VmDdxATfUkeR8Ulo9HmOeiVQOb/l9Hgi/gIa6UmH+/ahgX/B\
lEdVXcdXLakew7g/ZorTX38MC8abZKAkZprCILfaFDYS/IPqBr6+dYsiMxJYDEYZR7USqKpv\
m4Nx96+gAI4ZDFhhaaCD4fsWql3QcYKwlM9QjkFTALosMFBuMJUJ9o6Zxg6PlG+Y5EaCf1Dd\
YPt43a8OUWT80hmNfqNWCdQsbPu6FNYjuGmJ/t44PRuAN8pGNKIuGP8+j2HE2HzSUFJs8Yun\
LNWIXsIYvzCP317k9ztMRaylOpEGU7EvMQZOJ/jiQfroEqrnvRbOf7wuPiG58bhdJpfmPaz\
0/aIVQJhdbNUrhKIImNiNxr9R6USqFnYfjsa3ZJ8b5g2sqFVPYa/FhsNsFts/cOqxfO6HPnZ\
t7QfU5xQx8OaMBtv86+jaznLCXOfMW2Y07gq6laeASVytTue77VSvziQUnXtyyq2e8MPDmli\
f4cpypsdhQSF09z6QBWmDnOP4nHjz0UZjR2BlClyypmbyvD0Catbp+jMqVsUmeFZ+8NTqlGn\
BCpvb/kzPP/35IVLv02lfCKtkj9obazaYJLp+En5boTpPz0//4C28MdFbIESlklobJklBbr3\
IJX1vDeNhGX9ldfP61ZC/LSpofzvvf6Ou3V5lXu4vgMGznvKysq+hbwusKX4jhPuPUtpnefl\
c9x6WtNP0Tjhw/0cVDco8Q9XNVy2yluHKDLeNEaze8BKoHZhB57hI3DAyIbivvIb3DlXVrbE\
47EivNHDhwBoDG/YdvLaliXV69xp5HPd2lCFSQBxW/VtTY2iqPgf8cq9BhAebVoye2OOvJJf\
Nr+TdWy1O22n7siR8/H4zYNzt8F7kU9QllegEUxI3zFzVgLD2BFUNyhH37pFkRnGCAa9aANW\
AoNeogEkGDs39m00OONbsC9pzJOnVeqyliXVe/r8Ilw0L6tZDzH9Bje+xVGeE0KS/i16IQMq\
hzf9ICMYLAI53WWv/HB2B9UNPUBfJRBF2jgzGOyyjRolUFXfMR1Lb/8pDBDeFu3de3d9+dVH\
r/osLO5Aw/UKNeSnlyEbD4QXGwMjBIQZwYQ9cmcGwuqm7NyeQBSZCNhHtMioUQKWEA9jcUxg\
g8IQ4N3tO/Zf/vYTVw3JBzmJMcfEoQACVADMjELM0suYm+6v3TkYTlKYEaw73dXXE6i8vqVE\
HhWfLilxpmWJkzHjsDdtizXppFrT9sCcTQMtT2Vdy3jYTi7GtNlUSlpYuqwwQ6I+kra1Xlpq\
ferNlMb+LF8Oq5tIdfVzSl7FBlHtlDOAlYCKxbPDnzIhwJkxhhosNhn55++ZSi/yFvVMK8L\
NhM9bTgtuxwuFxQXvmN4vvJbLbUma78rduilZUnjIiE0wE2vLr90ilY6qgh4Phr/fOioSX2J\
QmPFsJJB/9XWd7xsiwM3NDfUfNQXnudFdX371VDKd0Ipnw8R6JeDh5XRh3hcMqslpIh/sain\
5pz2x7pTXT/U5QpLPqhu6FFt8FOkUWTCyjHawp37M6LrFY/FrwurAD7N/VlTQ2V7WLzBDsfD\
qS37wYeUF8aLilr0dGJwxHxCpdEmginGlbqByuKS9Wgci7IUgDdpKS6VIvFObf3KGd4gkxtv\
/nNq6ttbsC77llAAFyJe2POVkJa8qaSo7IOaug7jcutD+Znrhji+9gD0tYw8zDKHciyEq7Cb\
NBIYSEzzfS2ooBgG7Onq3p+XBT4onShhUthr8pKDQRMt4s2a+ra7pl/zVCIvGZ9I2gjm453x\
gs2KItNAhTjaFMftjzUJ4zEN+iO3n+FaVi/sWBq3il6HTKUhjtEbHZBS9Ep+WF3X9klJAQE\
1Q2dDF8lEEUmqAyjMeyId+UHChVLcmejexn8hrftxhWNFfWmvNAVH2snrONN4Sb/omRym18X\
lBlfGyxj2IMAb8ZMJ9gdZrxW6v20tOtaFs/BNw/5H9oIVnLOxLlhdoj8UzwYM5lS57UsLfdt\
ZDpGdX3HYksKI99888NSi25byfNaGma/65UJq5ut0nObG+a87JaLIuOWL5Trkd8TsETtUMC2\
ldOmG6rHx/GSonUJK7amv3+iuHizfnOb0tb++oHGGoLHg+LkhE15akzEnq9Z2PfferViTrjB\
I9QIZpAL87ZiMnfZcq9QTV37wvwUgOrE0ChwTQmGJyVQmL6rK8PqZnfmTn1GkQljMRrD40ds\
sU//ab4HI2TOmBlP1ZVB3Rk8eB81L52zypSdKi7+JuTNhjuTIPzlQ4v8b8Hl3QHRRPpA6nvx\
kng1JE4KiucNQ7muKykRlVUL2q9uWVL+mjfc6w4ygnnjOm60yxTqEWggx14JvuXW9gXU6V4n\
Le9Zv9mVLb8vkgeebr6/5o/ocU3BF51XoFfUAHhHeeMfdPtvshJUN9yDP7Y+VLXDm14UGW8a\
heAe0T2B3rFzSANWz+BGGt9AMChcOpAbjYaKGbDgAlb/j3u67TkohV5p2L9Dyql4E79SU99a\
Hi4YaATrE4diTKJ3tKjHTp/ZtLi8CNff6wv0uUD8KV5vvQYCPYD7jEMPpTbqrnlz4+x7tQLQ\
8prDisbyR3EzlnvTc9z4JmKcc519NtdNKmUYqkSRyc6lEFwjWglM/tzkacaHsPfu4QEOe4Oe\
PZAbjQc6L8Ofnnc/oOwKGClX9Dc/KJpxmFB7MWz2INAI5mSqxC5b2Jc0N5b/wNmVyF6d+hGC\
9zpRvGcYFPs+pnLCjh57xvXGng2+ycCipOv8xvZaXkrbbPhEJZw83OeguqHH4asEosi48yyU\
6xGtBLBXQKgxT8r0ZuPNvOYpJCEmGsPzCIAS+N88omWi6HOI8aHQpVAEt+JvX75ymXhSlCVk\
zNj1DlsZ15vXXtWjZrU0VKx0531wwY7a5fbLupYyS0HoHhg+0/5uVhy3Q6llK5bM7nB7Odd6\
jwdMVZpnAaR634nrnMPqZkdYKegn4+RXaOf4cFjsExW6UjF86x8sjeUv5sUuT89Lq/qOnQNR\
BOauqLFcCorggYvnv/w/pUWl9+C1eK0xpjdAiiuqbm+raVla0eQNCjOC6fi2UvXN95f/wSur\
3XhrTjKhRKP9OCOz5fgp8xD3RLefc42eF/oBSXd3X8L4OkEWJyqRzl9iCvGriGvsCSjb/rmT\
lnMOq5vfp9FRZJz8Cu0caBAa7jCwXdfn0bkMLGan2Bq4C3BaJC+Kq/gX/BLBW/4yPLT/4Bfm\
+CV7/LuiTrjp3LtC7rqaha0/haJ6xNi19iQA+8BX4JWjBIKMYJkklHihuaH8MU9yGScU0jRQ\
NIzFoSDS6bVuOdhR9DbpvkdmeBZPPFtb356AgpsI1TAR58z0aJjChiZ6ublxTos34cC6wfbg\
92l0FBlvvoXiHtFKAA/NNNPD6NzAouRx2qjloeP2nns/C/Y12GFj0AtDHtxtA11j37S4sgU/\
djLj2MllaKDyGm/5fNxn+vjBy2wE0/HTds9CfzkhSmKlXzSFwd/esbv7LVc4JhNkrcudcwnF\
ecjOEgLQEUYHYoJoOaB7RcrxO3Q21w2Rfe0BQTzMModyLKSrEW0TwKLUVNjNiiXEGWFxTOF4\
fo2r77QMxvWGB9CUor+//p5h97411yG9F/1juH2x3ZjPEWgEw94JQfsmwCJvrCca53vu7ylq\
FrTNQPbH+hQhkldm+IAl3du377/AtPAquG7+SiCKTKQKjAKhEa0E8Olozlzc3nsi7ZjhzemN\
metGj9fYOHTswXyjvP7QTUkM2u/MLUW2DxTTlGyf8D0FMRZ/0ivjdmPJrnF9vfRu1JHZYs0t\
HfFaiU/19xzpdPKspsbyv3ErGneKYUZB8MhRxFFk3HkW2vWIHg7gm4HthqFp333EJ2zlcBit\
6n0RPRflt7VNRrc3cEMQS6i8ZwY8yfs69c+LldR3dOHBLvWNcNBzhzcszAhmi+QzXhm3Gw19\
pmlY5VV0cE8NY+5Kuwe9m/lw70P6G9Fleh8K6X3M2LSl3lSr8vmMOKxuXV37+/35sJ+Mq8wF\
d3lYlMBgr0LEirW6V+4rX+K9O/hZrc157OL/FaxsuxJGsV975YPcibj5QxxHLpVK57yFnLAo\
5lnlLO4IUQC695Ez2xFkBEN3+5Og/RPL69qmQtnl9C6c8oNxVh2hAMxDAaWeSO1O3ZiR3SBS\
+TRyJx/TOahu6Itt6tlyLUs8ikxWAgXmGNHDgT1du1/F2+VA2D2DJesn+LXfvL6cc9JC4wke\
CujGtaxqsxPfOc+qf6pUYP2B4+7PuUyVXp5H/JwpPiUtYlnRaAN7K/gVI6MsymJv29X5ZnaZ\
5HHZ7kMujMM3tT5elZ35G6RfcQ6qm7eX4pQkiowjW4jnEa0EXn/oyvl4EIK/IMRdxdv1c2pM\
6S+xMejJ+d7koHGyTgONK+sNqf0qF3ZcVCam7aw9ftpGfPzzHblzj/bP56i+telEaVnLwuLC\
UPdsThxlHtObGogTRpBREAp2nc9YfZwj6z2DyQSv34DdAXXLsVc4mUWRcWQL8DyilYC+X9jn\
+6l87hsUQbUVT7yLMfe/YUoO2lz5Hxfe9sJR+odLkLC2JRgPJe0cJRBT6tu93Xk97353fEJ8\
DZTBTUH56QyqFrTNxe5CLOGlHGPMEAEYX+/bu2/P8+44YUYwKI3AngDKa+wJ4M2eU8fM2N5d\
ANc1FM6hqUGXf9TLsLrh24Sc8kWRiVq+OSJ3WGwCQ7kKce++tY8fM2a63mX4lNCborfyEuKu\
yZPLFmL7rPfRQNahsetlql114809FIzsJ4VVojMX6VR902GlP4zo4BLgkW0bq9B48dtKYH2Px\
zLN4aNcIqdZj0v4TK45xuLJORcO5BA2xOlvO4JLybt37cYeGGc4OJLtyGopbHuUzzgygl5Aj\
q6cMMbxyJ9F3jXrMqqlvu2pFQ8VzfZ6+F4usqrrqWTFLXgIjRwg/kPyvftHC6tbd3ZlTvigy\
fnkXkt9hUQJDCVBPrVUvbL8L+9fllSPQZcNDXIL/ZuA8w/08Y5FL3kWXVirrAayZMuVPIT/W\
N4HMZ7Pyb/HVXSZ3Ee8lGcCdf47ivY83bbnPm36gEUypruUF794UZBal0rjlBpe2lmU0IvQXp\
c8v/Rm9rqS3Vz2P7ujd0pfbbpUdPmJyS8tiYrWYC8ZegXKrBPvNlIpQg/glfJRBUNyjszf01\
Cppk+opeoBcjfjig7lvz4vKncYOXDtU9xNvw45wNOK3Mz4QfniLgy79UKnnNu0/P6/FmEGIB\
CxwKhBkFU8nkam9+6bT4vV7g4/Xvc+velhR3xIRcLcaW7ikdP/FTGGXW422zCvsJLsfltY4C\
0DJQgp/lyXougUoGySwl7IhGkXFkC/U8KpSAvnlNDU3leKkYdxAa3BssV+akl8fqxRyZfDyw\
qCYlknNbl1a94xs9wAiGxurbUJx0woyCfrsfty2f8yHk3B8IOclFOuvhhVEwoG5QQ/5liyJj\
```

```
LEBhBIwaJYBf57JXbNr6Nbyi7sLDnzxctw9pd9kpdY83/T2de9rQGxncXxJSYhUUwEx83+D7\
RtdGMLx1T/WWxXGHLWaCbP+Mgr0Jd6ktd4LDOiefAZ2lgkE09wirm23lrtGIIpObc+H5jCIl\
gJuHT4ObFs/+d/zA6AV4w7yCB9Xcbe3/vd4mlL08mUxP99t0Uxvs0LWdCUXQ2v+ksyVQ7I/x\
vfsdTZu3lPd+4JQdoddVPHb8DFwa72GqJ7ehuBNCeY3WfD+joCOrf08BjP8abv/eiRMx5Iz7\
81HX/i7fXkVY3ax0MqcnEEUmpIgFETziDYN+dwkr5PRY9lL968ExFb8ZA8/LMQ49HX7GBuNN\
Bw2xG/PQq6FFXhMy/VzTpmltWsl447ndsHL/H9xVNQtaz5dW7BaMf+fCPc0dx3iNRU/IC3sh\
ysc/3rzlSb/xv1dWpq0dKq52oG6T3GEoewqbnv1Cb+fl9vdeI781UAQ5S6Mhv0Yc6PmVN77b\
fZDxorNr6qvn4Vd//xlhmm/ogYaPXprCIi/x63RP6iEY9/b5CeHLsO0JpbaD4WR3uK4b7suT\
TUtqP3H76+soMt40CtGNZ6AwDtlVLBl/9HlKxaejm4z99MV41Bx/ejpQ7cVU4U6cd+Jb+R22\
st/bu3/t7zMf9QwQT+X8ltOteOwi/AzXFPQkjsNDrZfdQvnKXTo/W8gdIpl+Y6Pd+doHy644\
ECE7CSv/lJhMj4n1xK2ksru2xDq35ZuWXknZk0hMsIRdjFJ1I/89/f+B1EVWxelVJydi1olo\
3Cfgm44T0dCnYFuyHtR3Lz4UgnFQbLSlXKt2pdbpFYV51lNic9Lj8GuNY6WtYulkbH8edYsi\
k2dxGI0ESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAE\
SIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAE\
SIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAE\
SIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAE\
SIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAE\
SIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAESIAE\
AABJRU5ErkJggg==";
document.getElementById('banner').style.backgroundImage="url("+GshLogo+")";
document.getElementById('gsh-footer').style.backgroundImage="url("+"QR-ITS-more.jp.png"+")";
//https://www.w3schools.com/JSREF/prop_style_backgroundposition.asp
var bannerStop = false
function shiftBG(){
 bannerStop = !bannerStop
 document.getElementById('banner').style.backgroundPosition = "100 0";
}
//https://www.w3schools.com/jsref/met_win_setinterval.asp
function shiftBanner(){
 var now = new Date().getTime();
 //"console.log("now="+(now%10))
 if( !bannerStop ){
  document.getElementById('banner').style.backgroundPosition = ((now/10)%100000)+" 0";
 }
}
setInterval(shiftBanner,10);
</script>
-->
*/ //</span></html>
```