

//*

GShell version 0.1.7 // 2020-08-21 // SatoxITS



GShell // a General purpose Shell built on the top of Golang

► Overview

▼ Index

Implementation

Structures

[import](#)[struct](#)

Main functions

[str-expansion](#) // macro processor
[finder](#) // builtin find + du
[grep](#) // builtin grep + wc + cksum + ...
[plugin](#) // plugin commands
[system](#) // external commands
[builtin](#) // builtin commands
[network](#) // socket handler
[remote-sh](#) // remote shell
[redirect](#) // StdIn/Out redirection
[history](#) // command history
[rusage](#) // resource usage
[encode](#) // encode / decode
[getline](#) // line editor
[scanf](#) // string decomposer
[interpreter](#) // command interpreter
[main](#)

▼ Source Code

```
// gsh - Go lang based Shell
// (c) 2020 ITS more Co., Ltd.
// 2020-0807 created by SatoxITS (sato@its-more.jp)

package main // gsh main
// Imported packages // Packages
import (
    "fmt" // fmt
    "strings" // strings
    "strconv" // strconv
    "sort" // sort
    "time" // time
    "bufio" // bufio
    "io/ioutil" // ioutil
    "os" // os
    "syscall" // syscall
    "plugin" // plugin
    "net" // net
    "net/http" // http
    "html" // html
    "path/filepath" // filepath
    "go/types" // types
    "go/token" // token
    "encoding/base64" // base64
    //"gshdata" // gshell's logo and source code
)

var NAME = "gsh"
var VERSION = "0.1.7"
var DATE = "2020-0821"
var LINESIZE = (8*1024)
var PATHSEP = ";" // should be ";" in Windows
var DIRSEP = "/" // canbe \ in Windows
var GSH_HOME = ".gsh" // under home directory
var MaxStreamSize = int64(128*1024*1024*1024) // 128GiB is too large?
var PROMPT = "> "

// -x logging control
// --A-- all
// --I-- info.
// --D-- debug
// --T-- time and resource usage
// --W-- warning
// --E-- error
// --F-- fatal error
// --Xn- network

// Structures
type GCommandHistory struct {
    StartAt time.Time // command line execution started at
    EndAt time.Time // command line execution ended at
    ResCode int // exit code of (external command)
    CmdError error // error string
    OutData *os.File // output of the command
    FoundFile []string // output - result of ufind
    Rusagev [2]syscall.Rusage // Resource consumption, CPU time or so
    CmdId int // maybe with identified with arguments or impact
    // redirection commands should not be the CmdId
    WorkDir string // working directory at start
    WorkDirX int // index in ChdirHistory
    CmdLine string // command line
}
type GChdirHistory struct {
    Dir string
    MovedAt time.Time
    CmdIndex int
}
type CmdMode struct {
    BackGround bool
}
type PluginInfo struct {
    Spec *plugin.Plugin
    Addr plugin.Symbol
    Name string // maybe relative
    Path string // this is in Plugin but hidden
}
type GServer struct {
    host string
    port string
}
```

```

}
type ValueStack [][]string
type GshContext struct {
    StartDir      string // the current directory at the start
    GetLine       string // gsh-getline command as a input line editor
    CkdirHistory  []GchdirHistory // the 1st entry is wd at the start
    gshPA         syscall.ProcAttr
    CommandHistory []GCommandHistory
    CmdCurrent    GCommandHistory
    Background    bool
    BackgroundJobs []int
    LastRusage    syscall.Rusage
    GshHomeDir    string
    TerminalId    int
    CmdTrace      bool // should be [map]
    CmdTime       bool // should be [map]
    PluginFuncs   []PluginInfo
    iValues       []string
    idelimiter    string // field separator of print out
    iFormat       string // default print format (of integer)
    iValStack     ValueStack
    LastServer    GServer
}

func strBegins(str, pat string)(bool){
    if len(pat) <= len(str){
        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 symbol
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, rstr 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 {
        var ext byte
        if 1 < len(str[i+leng:]) {
            ext = str[i+leng:][1]
        }
        //fmt.Printf("--D-- %v(%c)\n",str[i+leng:],str[i+leng])
        if ext == '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{
            if ext == '@' || ext == 'd' {
                // !N@ .. workdir at the start of the command
                leng += 1
                rstr += gshCtx.CommandHistory[hi].WorkDir
            }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
}
}
// str-expansion
// - this should be a macro processor
func strsubst(gshCtx *GshContext,str string,histonly bool) string {
rbuff := []byte{}
if false {
//@@@U Unicode should be cared as a character
return str
}
//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)
leng,rs := substHistory(gshCtx,str,i,"")
if 0 < leng {
//_,rs := substHistory(gshCtx,str,i,"")
rbuff = append(rbuff,[]byte(rs)...)
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)
rs := time.Now().Format(time.Stamp)
rbuff = append(rbuff,[]byte(rs)...)
inEsc = 0
continue;
default:
// postpone the interpretation
//rstr = rstr + "%" + string(ch)
rbuff = append(rbuff,ch)
inEsc = 0
continue;
}
inEsc = 0
}
//rstr = rstr + string(ch)
rbuff = append(rbuff,ch)
}
//fmt.Printf("--D--subst(%s)(%s)\n",str,string(rbuff))
return string(rbuff)
//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
}

// Encode / Decode
// Encoder
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
}

// lnspl [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-- lnspl %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("\\"))
os.Stdout.Write([]byte("\n"))
}
line = line[olen:]
off += olen
remlen -= olen
}
}
fmt.Fprintf(os.Stderr,"--I-- lnspl %d to %d\n",ni,toi)
return gshCtx
}

// grep
// "lines", "lin" or "lnp" for "(text) line processor" or "scanner"
// a*,!ab,c, ... sequential 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 (gsh*GshContext)xGrep(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 gsh.CmdTrace { fmt.Printf("--I-- grep %v %v\n", path, rexpv) }
    //reader := bufio.NewReaderSize(file, LINE_SIZE)
    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
}

// Finder
// 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(){
            tstamp := now.Format(time.Stamp)
            showFusage(tstamp, fsum)
        }
        fsum.nextRepo = now.Add(time.Second)
    }
    if staterr != nil {
        fsum.Err += 1
        return fsum
    }
    fsum.Files += 1
    if l < 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 (gsh*GshContext)xxFindEnvf(depth int,total *fileSum,dir string, dstat syscall.Stat_t, ei int, envt []string,npatv[]string,argv[]string)(*fileSum){
    nols := isin("-grep",argv)
    // sort envt
    /*
    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 envt {
        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 = cumPinfo(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 := gsh.xGrep(path,argv[x+1:])
                    if 0 < found {
                        foundv := gsh.CmdCurrent.FoundFile
                        if len(foundv) < 10 {
                            gsh.CmdCurrent.FoundFile =
                                append(gsh.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 {
                    total = gsh.xxFind(depth+1,total,path,npatv,argv)
                }
            }
        }
    }
    return total
}
func (gsh*GshContext)xxFind(depth int,total *fileSum,dir string,npatv[]string,argv[]string)(*fileSum){
    nols := isin("-grep",argv)
    dirfile,oerr := os.OpenFile(dir,os.O_RDONLY,0)
    if oerr == nil {
        //fmt.Printf("--I-- %v(%v)[%d]\n",dir,dirfile,dirfile.Fd())
        defer dirfile.Close()
    }else{
    }
    prev := *total
    var dstat syscall.Stat_t
    staterr := syscall.Lstat(dir,&dstat) // should be flstat
    if staterr != nil {
        if !isin("-w",argv){ fmt.Printf("ufind: %v\n",staterr) }
        return total
    }
    //filev,err := ioutil.ReadDir(dir)
    //_,err := ioutil.ReadDir(dir) // ReadDir() heavy and bad for huge directory
    /*
    if err != nil {
        if !isin("-w",argv){ fmt.Printf("ufind: %v\n",err) }
        return total
    }
    */
    if depth == 0 {

```

```

        total = cumFinfo(total,dir,staterr,dstat,argv,true)
        if !nols && !isin("-s",argv) && (!isin("-du",argv) || isin("-a",argv)) {
            showFileInfo(dir,argv)
        }
    }
    // it it is not a directory, just scan it and finish
    for ei := 0; ; ei++ {
        entv,rderr := dirfile.Readdirnames(8*1024)
        if len(entv) == 0 || rderr != nil {
            //if rderr != nil { fmt.Printf("[%d] len=%d (%v)\n",ei,len(entv),rderr) }
            break
        }
        if 0 < ei {
            fmt.Printf("--I-- xxFind[%d] %d large-dir: %s\n",ei,len(entv),dir)
        }
        total = gsh.xxFindEntv(depth,total,dir,dstat,ei,entv,npatv,argv)
    }
    if isin("-du",argv) {
        // if in "du" mode
        fmt.Printf("%d\t%s\n",total.Blocks-prev.Blocks)/2,dir)
    }
    return total
}

// {ufind|fu|ls} [Files] [Names] [-- Expressions]
// Files is "." by default
// Names is "*" by default
// Expressions is "-print" by default for "ufind", or -du for "fu" command
func (gsh*GshContext)xFind(argv[]string){
    if 0 < len(argv) && strBegins(argv[0],"?"){
        showFound(gsh,argv)
        return
    }
    var total = fileSum{}
    npats := []string{}
    for _,v := range argv {
        if 0 < len(v) && v[0] != '-' {
            npats = append(npats,v)
        }
        if v == "/" { break }
        if v == "--" { break }
        if v == "-grep" { break }
        if v == "-ls" { break }
    }
    if len(npats) == 0 {
        npats = []string{"*"}
    }
    cwd := "."
    // if to be fullpath :: cwd, _ := os.Getwd()
    if len(npats) == 0 { npats = []string{"*"} }
    fusage := gsh.xxFind(0,&total,cwd,npats,argv)
    if !isin("-grep",argv) {
        showFusage("total",fusage)
    }
    if !isin("-s",argv){
        hits := len(gsh.CmdCurrent.FoundFile)
        if 0 < hits {
            fmt.Printf("--I-- %d files hits // can be refered with !%df\n",
                hits,len(gsh.CommandHistory))
        }
    }
    return
}

func showFiles(files[]string){
    sp := ""
    for i,file := range files {
        if 0 < i { sp = " " } else { sp = "" }
        fmt.Printf(sp+"%s",escapeWhiteSP(file))
    }
}

func showFound(gshCtx *GshContext, argv[]string){
    for i,v := range gshCtx.CommandHistory {
        if 0 < len(v.FoundFile) {
            fmt.Printf("!%d (%d) ",i,len(v.FoundFile))
            if isin("-ls",argv){
                fmt.Printf("\n")
                for _,file := range v.FoundFile {
                    fmt.Printf("%s") //sub number?
                    showFileInfo(file,argv)
                }
            }else{
                showFiles(v.FoundFile)
                fmt.Printf("\n")
            }
        }
    }
}

func showMatchFile(filev []os.FileInfo, npat,dir string, argv[]string)(string,bool){
    fname := ""
    found := false
    for _,v := range filev {
        match, _ := filepath.Match(npat,(v.Name()))
        if match {
            fname = v.Name()
            found = true
            //fmt.Printf("[%d] %s\n",i,v.Name())
            showIfExecutable(fname,dir,argv)
        }
    }
    return fname,found
}

func showIfExecutable(name,dir string,argv[]string)(ffullpath string,ffound bool){
    var fullpath string
    if strBegins(name,DIRSEP){
        fullpath = name
    }else{
        fullpath = dir + DIRSEP + name
    }
    fi, err := os.Stat(fullpath)
    if err != nil {
        fullpath = dir + DIRSEP + name + ".go"
        fi, err = os.Stat(fullpath)
    }
    if err == nil {
        fm := fi.Mode()
        if fm.IsRegular() {
            // R_OK=4, W_OK=2, X_OK=1, F_OK=0
            if syscall.Access(fullpath,5) == nil {
                ffullpath = fullpath
                ffound = true
                if ! isin("-s", argv) {
                    showFileInfo(fullpath,argv)
                }
            }
        }
    }
}

```

```

    }
    }
}
return ffullpath, ffound
}
func which(list string, argv []string) (fullpathv []string, itis bool){
if len(argv) <= 1 {
    fmt.Printf("Usage: which comand [-s] [-a] [-ls]\n")
    return []string{"", false}
}
path := argv[1]
if strBegins(path, "/") {
    // should check if excecutable?
    _, exOK := showIfExecutable(path, "/", argv)
    fmt.Printf("--D-- %v exOK=%v\n", path, exOK)
    return []string{path}, exOK
}
pathenv, efound := os.LookupEnv(list)
if ! efound {
    fmt.Printf("--E-- which: no \"%s\" environment\n", list)
    return []string{"", false}
}
showall := isin("-a", argv) || 0 <= strings.Index(path, "*")
dirv := strings.Split(pathenv, PATHSEP)
ffound := false
ffullpath := path
for _, dir := range dirv {
    if 0 <= strings.Index(path, "*") { // by wild-card
        list, _ := ioutil.ReadDir(dir)
        ffullpath, ffound = showMatchFile(list, path, dir, argv)
    } else {
        ffullpath, ffound = showIfExecutable(path, dir, argv)
    }
    //if ffound && !isin("-a", argv) {
    //if ffound && !showall {
        break;
    }
}
return []string{ffullpath}, ffound
}
func stripLeadingWSParg(argv []string) ([]string){
for ; 0 < len(argv); {
    if len(argv[0]) == 0 {
        argv = argv[1:]
    } else {
        break
    }
}
return argv
}
func xEval(argv []string, nlend bool){
argv = stripLeadingWSParg(argv)
if len(argv) == 0 {
    fmt.Printf("eval [%%format] [Go-expression]\n")
    return
}
pfmt := "%v"
if argv[0][0] == '%' {
    pfmt = argv[0]
    argv = argv[1:]
}
if len(argv) == 0 {
    return
}
gocode := strings.Join(argv, " ");
//fmt.Printf("eval [%v] [%v]\n", pfmt, gocode)
fset := token.NewFileSet()
rval, _ := types.Eval(fset, nil, token.NoPos, gocode)
fmt.Printf(pfmt, rval.Value)
if nlend { fmt.Printf("\n") }
}
func getval(name string) (found bool, val int) {
/* should expand the name here */
if name == "gsh.pid" {
    return true, os.Getpid()
} else {
if name == "gsh.ppid" {
    return true, os.Getppid()
}
}
return false, 0
}
func echo(argv []string, nlend bool){
for ai := 1; ai < len(argv); ai++ {
    if 1 < ai {
        fmt.Printf(" ");
    }
    arg := argv[ai]
    found, val := getval(arg)
    if found {
        fmt.Printf("%d", val)
    } else {
        fmt.Printf("%s", arg)
    }
}
if nlend {
    fmt.Printf("\n");
}
}
func resfile() string {
return "gsh.tmp"
}
//var resF *File
func resmap() {
//_, err := os.OpenFile(resfile(), os.O_RDWR|os.O_CREATE, os.ModeAppend)
// https://developpaper.com/solution-to-golang-bad-file-descriptor-problem/
_, err := os.OpenFile(resfile(), os.O_RDWR|os.O_CREATE, 0600)
if err != nil {
    fmt.Printf("refF could not open: %s\n", err)
} else {
    fmt.Printf("refF opened\n")
}
}
// @@2020-0821
func gshScanArg(str string, strip int)(argv []string){
var si = 0
var sb = 0
var inBracket = 0
var arg1 = make([]byte, LINESIZE)
var ax = 0

```



```

debug := false
for ; si < len(str); si++ {
    if str[si] != ' ' {
        break
    }
}
sb = si
for ; si < len(str); si++ {
    if sb <= si {
        if debug {
            fmt.Printf("--Da- +%d %2d-%2d %s ... %s\n",
                inBracket,sb,si,argv[0:ax],str[si:])
        }
        ch := str[si]
        if ch == '{' {
            inBracket += 1
            if 0 < strip && inBracket <= strip {
                //fmt.Printf("stripLEV %d <= %d?\n",inBracket,strip)
                continue
            }
        }
        if 0 < inBracket {
            if ch == '}' {
                inBracket -= 1
                if 0 < strip && inBracket < strip {
                    //fmt.Printf("stripLEV %d < %d?\n",inBracket,strip)
                    continue
                }
            }
            argl[ax] = ch
            ax += 1
            continue
        }
        if str[si] == ' ' {
            argv = append(argv,string(argv[0:ax]))
            if debug {
                fmt.Printf("--Da- [%v][%v-%v] %s ... %s\n",
                    -1+len(argv),sb,si,str[sb:si],string(str[si:]))
            }
            sb = si+1
            ax = 0
            continue
        }
        argl[ax] = ch
        ax += 1
    }
    if sb < si {
        argv = append(argv,string(argv[0:ax]))
        if debug {
            fmt.Printf("--Da- [%v][%v-%v] %s ... %s\n",
                -1+len(argv),sb,si,string(argv[0:ax]),string(str[si:]))
        }
    }
    if debug {
        fmt.Printf("--Da- %d [%s] => [%d]%v\n",strip,si,len(argv),argv)
    }
    return argv
}

// should get stderr (into tmpfile ?) and return
func (gsh*GshContext)Popen(name,mode string)(pin*os.File,pout*os.File,err bool){
    var pv = [2]int{-1,-1}
    syscall.Pipe(pv)

    xarg := gshScanArg(name,1)
    name = strings.Join(xarg," ")

    pin = os.NewFile(uintptr(pv[0]),"StdoutOf-"+name)
    pout = os.NewFile(uintptr(pv[1]),"StdinOf-"+name)
    fdix := 0
    dir := "?"
    if mode == "r" {
        dir = "<"
        fdix = 1 // read from the stdout of the process
    }else{
        dir = ">"
        fdix = 0 // write to the stdin of the process
    }
    gshPA := gsh.gshPA
    savfd := gshPA.Files[fdix]

    var fd uintptr = 0
    if mode == "r" {
        fd = pout.Fd()
        gshPA.Files[fdix] = pout.Fd()
    }else{
        fd = pin.Fd()
        gshPA.Files[fdix] = pin.Fd()
    }
    fmt.Printf("--Ip- Opened fd[%v] %s %v\n",fd,dir,name)
    // should do this by Goroutine?
    gsh.BackGround = true
    gshell1(*gsh,name)
    gsh.BackGround = false

    gshPA.Files[fdix] = savfd
    return pin,pout,false
}

// External commands
func (gsh*GshContext)excommand(exec bool, argv []string) (notf bool,exit bool) {
    if gsh.CmdTrace { fmt.Printf("--I-- excommand[%v](%v)\n",exec,argv) }

    gshPA := gsh.gshPA
    fullpath, itis := which("PATH",[]string{"which",argv[0],"-s"})
    if itis == false {
        return true,false
    }
    fullpath := fullpath[0]
    argv = unescapeWhiteSPV(argv)
    if 0 < strings.Index(fullpath,".go") {
        nargv := argv // []string{}
        gofullpath, itis := which("PATH",[]string{"which","go","-s"})
        if itis == false {
            fmt.Printf("--F-- Go not found\n")
            return false,true
        }
        gofullpath := gofullpath[0]
        nargv = []string{ gofullpath, "run", fullpath }
        fmt.Printf("--I-- %s {%s %s %s}\n",gofullpath,
            nargv[0],nargv[1],nargv[2])
        if exec {

```

```

    }else{
        syscall.Exec(gofullpath,nargv,os.Environ())
    }else{
        pid, _ := syscall.ForkExec(gofullpath,nargv,&gshPA)
        if gsh.Background {
            fmt.Printf("--Ip- in Background pid[%d]\n",pid)
            gsh.BackGroundJobs = append(gsh.BackGroundJobs,pid)
        }else{
            rusage := syscall.Rusage {}
            syscall.Wait4(pid,nil,0,&rusage)
            gsh.LastRusage = rusage
            gsh.CmdCurrent.Rusagev[1] = rusage
        }
    }
}
}
}
return false,false
}

// Builtin Commands
func sleep(gshCtx GshContext, argv []string) {
    if len(argv) < 2 {
        fmt.Printf("Sleep 100ms, 100us, 100ns, ... \n")
        return
    }
    duration := argv[1];
    d, err := time.ParseDuration(duration)
    if err != nil {
        d, err = time.ParseDuration(duration+"s")
        if err != nil {
            fmt.Printf("duration ? %s (%s)\n",duration,err)
            return
        }
    }
    //fmt.Printf("Sleep %v\n",duration)
    time.Sleep(d)
    if 0 < len(argv[2:]) {
        gshellv(gshCtx, argv[2:])
    }
}

func repeat(gshCtx GshContext, argv []string) {
    if len(argv) < 2 {
        return
    }
    start0 := time.Now()
    for ri, _ := strconv.Atoi(argv[1]); 0 < ri; ri-- {
        if 0 < len(argv[2:]) {
            //start := time.Now()
            gshellv(gshCtx, argv[2:])
            end := time.Now()
            elps := end.Sub(start0);
            if( 1000000000 < elps ){
                fmt.Printf("repeat#%d %v\n",ri,elps);
            }
        }
    }
}

func gen(gshCtx GshContext, argv []string) {
    gshPA := gshCtx.gshPA
    if len(argv) < 2 {
        fmt.Printf("Usage: %s N\n",argv[0])
        return
    }
    // should br repeated by "repeat" command
    count, _ := strconv.Atoi(argv[1])
    fd := gshPA.Files[1] // Stdout
    file := os.NewFile(fd,"internalStdOut")
    fmt.Printf("--I-- Gen. Count=%d to [%d]\n",count,file.Fd())
    //buf := []byte{}
    outdata := "0123 5678 0123 5678 0123 5678 0123 5678\r"
    for gi := 0; gi < count; gi++ {
        file.WriteString(outdata)
    }
    //file.WriteString("\n")
    fmt.Printf("\n(%d B)\n",count*len(outdata));
    //file.Close()
}

// Remote Execution // 2020-0820
func Elapsed(from time.Time)(string){
    elps := time.Now().Sub(from)
    if 1000000000 < elps {
        return fmt.Sprintf("[%5d.%02ds]",elps/1000000000,(elps%1000000000)/10000000)
    }else
    if 1000000 < elps {
        return fmt.Sprintf("[%3d.%03dms]",elps/1000000,(elps%1000000)/1000)
    }else{
        return fmt.Sprintf("[%3d.%03dus]",elps/1000,(elps%1000))
    }
}

func absize(size int64)(string){
    fsize := float64(size)
    if 1024*1024*1024 < size {
        return fmt.Sprintf("%8.2fGiB",fsize/(1024*1024*1024))
    }else
    if 1024*1024 < size {
        return fmt.Sprintf("%8.3fMiB",fsize/(1024*1024))
    }else{
        return fmt.Sprintf("%8.3fKiB",fsize/1024)
    }
}

func abspeed(totalB int64,ns time.Duration)(string){
    MBs := (float64(totalB)/1000000) / (float64(ns)/1000000000)
    if 1000 <= MBs {
        return fmt.Sprintf("%6.3fGBps",MBs/1000)
    }
    if 1 <= MBs {
        return fmt.Sprintf("%6.3fMBps",MBs)
    }else{

```

```

        return fmt.Sprintf("%.3fKBps",MBs*1000)
    }
}
func fileRelay(what string,in*os.File,out*os.File,size int64,bsiz int)(wcount int64){
    Start := time.Now()
    buff := make([]byte,bsiz)
    var total int64 = 0
    var rem int64 = size
    nio := 0
    Prev := time.Now()
    var PrevSize int64 = 0

    fmt.Printf(Elapsed(Start)+"--In- X: %s (%v/%v/%v) START\n",
        what,absize(total),size,nio)

    for i:= 0; ; i++ {
        var len = bsiz
        if int(rem) < len {
            len = int(rem)
        }
        Now := time.Now()
        Elps := Now.Sub(Prev);
        if 1000000000 < Now.Sub(Prev) {
            fmt.Printf(Elapsed(Start)+"--In- X: %s (%v/%v/%v) %s\n",
                what,absize(total),size,nio,
                abspeed((total-PrevSize),Elps))
            Prev = Now;
            PrevSize = total
        }
        rlen := len
        if in != nil {
            // should watch the disconnection of out
            rcc,err := in.Read(buff[0:rlen])
            if err != nil {
                fmt.Printf(Elapsed(Start)+"--En- X: %s read(%v,%v)<%v\n",
                    what,rcc,err,in.Name())
                break
            }
            rlen = rcc
            if string(buff[0:10]) == "(SoftEOF " {
                var ecc int64 = 0
                fmt.Sscanf(string(buff),"(SoftEOF %v",&ecc)
                fmt.Printf(Elapsed(Start)+"--En- X: %s Recv ((SoftEOF %v))/%v\n",
                    what,ecc,total)
                if ecc == total {
                    break
                }
            }
        }
        wlen := rlen
        if out != nil {
            wcc,err := out.Write(buff[0:rlen])
            if err != nil {
                fmt.Printf(Elapsed(Start)+"--En-- X: %s write(%v,%v)>%v\n",
                    what,wcc,err,out.Name())
                break
            }
            wlen = wcc
        }
        if wlen < rlen {
            fmt.Printf(Elapsed(Start)+"--En- X: %s incomplete write (%v/%v)\n",
                what,wlen,rlen)
            break;
        }

        nio += 1
        total += int64(rlen)
        rem -= int64(rlen)
        if rem <= 0 {
            break
        }
    }
    Done := time.Now()
    Elps := float64(Done.Sub(Start))/1000000000 //Seconds
    TotalMB := float64(total)/1000000 //MB
    MBps := TotalMB / Elps
    fmt.Printf(Elapsed(Start)+"--In- X: %s (%v/%v/%v) %v %.3fMB/s\n",
        what,total,size,nio,absize(total),MBps)
    return total
}
func (gsh*GshContext)RexecServer(argv[]string){
    debug := true
    Start0 := time.Now()
    Start := Start0
    // if local == ":" { local = "0.0.0.0:9999" }
    local := "0.0.0.0:9999"

    if 0 < len(argv) {
        if argv[0] == "-s" {
            debug = false
            argv = argv[1:]
        }
    }
    if 0 < len(argv) {
        argv = argv[1:]
    }
    port, err := net.ResolveTCPAddr("tcp",local);
    if err != nil {
        fmt.Printf("--En- S: Address error: %s (%s)\n",local,err)
        return
    }
    fmt.Printf(Elapsed(Start)+"--In- S: Listening at %s...\n",local);
    sconn, err := net.ListenTCP("tcp", port)
    if err != nil {
        fmt.Printf(Elapsed(Start)+"--En- S: Listen error: %s (%s)\n",local,err)
        return
    }

    reqbuf := make([]byte,LINESIZE)
    res := ""
    for {
        fmt.Printf(Elapsed(Start0)+"--In- S: Accepting at %s...\n",local);
        aconn, err := sconn.AcceptTCP()
        Start = time.Now()
        if err != nil {
            fmt.Printf(Elapsed(Start)+"--En- S: Accept error: %s (%s)\n",local,err)
            return
        }
        clnt, _ := aconn.File()
        fd := Clnt.Fd()
        if debug { fmt.Printf(Elapsed(Start0)+"--In- S: Accepted TCP at %s [%d]\n",local,fd) }
        res = fmt.Sprintf("220 GShell/%s Server\r\n",VERSION)
        fmt.Fprintf(clnt,"%s",res)
        if debug { fmt.Printf(Elapsed(Start)+"--In- S: %s",res) }
        count, err := clnt.Read(reqbuf)
    }
}

```

```

if err != nil {
    fmt.Printf(Elapsed(Start)+"--En- C: (%v %v) %v",
        count,err,string(reqbuf))
}
req := string(reqbuf[:count])
if debug { fmt.Printf(Elapsed(Start)+"--In- C: %v",string(req)) }
reqv := strings.Split(string(req),"\r")
cmdv := gshScanArg(reqv[0],0)
//cmdv := strings.Split(reqv[0], " ")
switch cmdv[0] {
case "HELO":
    res = fmt.Sprintf("250 %v",req)
case "GET":
    // download {remotefile|-zN} [localfile]
    var dsize int64 = 32*1024*1024
    var bsize int = 64*1024
    var fname string = ""
    var in *os.File = nil
    var pseudoEOF = false
    if 1 < len(cmdv) {
        fname = cmdv[1]
        if strBegins(fname,"-z") {
            fmt.Sscanf(fname[2:], "%d",&dsize)
        }else
        if strBegins(fname,"") {
            xin,xout,err := gsh.Popen(fname,"r")
            if err {
            }else{
                xout.Close()
                defer xin.Close()
                in = xin
                dsize = MaxStreamSize
                pseudoEOF = true
            }
        }else{
            xin,err := os.Open(fname)
            if err != nil {
                fmt.Printf("--En- GET (%v)\n",err)
            }else{
                defer xin.Close()
                in = xin
                fi, _ := xin.Stat()
                dsize = fi.Size()
            }
        }
    }
    //fmt.Printf(Elapsed(Start)+"--In- GET %v:%v\n",dsize,bsize)
    res = fmt.Sprintf("200 %v\r\n",dsize)
    fmt.Fprintf(clnt,"%v",res)
    fmt.Printf(Elapsed(Start)+"--In- S: %v",res)
    wcount := fileRelay("SendGET",in,clnt,dsize,bsize)
    if pseudoEOF {
        // show end of stream data (its size) by OOB?
        time.Sleep(100*1000*1000)
        SoftEOF := fmt.Sprintf("((SoftEOF %v))",wcount)
        fmt.Printf(Elapsed(Start)+"--In- S: Send %v\n",SoftEOF)
        fmt.Fprintf(clnt,"%v\r\n",SoftEOF)
        // with client generated random?
    }
    res = fmt.Sprintf("200 GET done\r\n")
case "PUT":
    // upload {srcfile|-zN} [dstfile]
    var dsize int64 = 32*1024*1024
    var bsize int = 64*1024
    var fname string = ""
    var out *os.File = nil
    if 1 < len(cmdv) { // localfile
        fmt.Sscanf(cmdv[1],"%d",&dsize)
    }
    if 2 < len(cmdv) {
        fname = cmdv[2]
        if fname == "-" {
            // nul dev
        }else
        if strBegins(fname,"") {
            xin,xout,err := gsh.Popen(fname,"w")
            if err {
            }else{
                xin.Close()
                defer xout.Close()
                out = xout
            }
        }else{
            // should write to temporary file
            // should suppress ^C on tty
            xout,err := os.OpenFile(fname,os.O_CREATE|os.O_RDWR|os.O_TRUNC,0600)
            //fmt.Printf("--In- S: open(%v) out(%v) err(%v)\n",fname,xout,err)
            if err != nil {
                fmt.Printf("--En- PUT (%v)\n",err)
            }else{
                out = xout
            }
        }
    }
    fmt.Printf(Elapsed(Start)+"--In- L: open(%v,w) %v (%v)\n",
        fname,local,err)
    }
    fmt.Printf(Elapsed(Start)+"--In- PUT %v (%v)\n",dsize,bsize)
    fmt.Printf(Elapsed(Start)+"--In- S: 200 %v OK\r\n",dsize)
    fmt.Fprintf(clnt,"200 %v OK\r\n",dsize)
    fileRelay("RecvPUT",clnt,out,dsize,bsize)
    res = fmt.Sprintf("200 PUT done\r\n")
default:
    res = fmt.Sprintf("400 What? %v",req)
}
clnt.Write([]byte(res))
fmt.Printf(Elapsed(Start)+"--In- S: %v",res)
acconn.Close();
clnt.Close();
}
sconn.Close();
}
func (gsh*GshContext)RexecClient(argv[]string){
    debug := true
    Start := time.Now()
    if len(argv) == 1 {
        return
    }
    argv = argv[1:]
    if argv[0] == "-serv" {
        gsh.RexecServer(argv[1:])
        return
    }
    remote := "0.0.0.0:9999"
    if argv[0][0] == '@' {
        remote = argv[0][1:]
    }
}

```

```

    argv = argv[1:]
}
if argv[0] == "-s" {
    debug = false
    argv = argv[1:]
}
dport, err := net.ResolveTCPAddr("tcp", remote);
if err != nil {
    fmt.Printf(Elapsed(Start)+"Address error: %s (%s)\n", remote, err)
    return
}
fmt.Printf(Elapsed(Start)+"--In- C: Socket: connecting to %s\n", remote)
serv, err := net.DialTCP("tcp", nil, dport)
if err != nil {
    fmt.Printf(Elapsed(Start)+"Connection error: %s (%s)\n", remote, err)
    return
}
if debug { fmt.Printf(Elapsed(Start)+"--In- C: Socket: connected to %s\n", remote) }

req := ""
res := make([]byte, LINESIZE)
count, err := serv.Read(res)
if err != nil {
    fmt.Printf("--En- S: (%3d,%v) %v", count, err, string(res))
}
if debug { fmt.Printf(Elapsed(Start)+"--In- S: %v", string(res)) }

if argv[0] == "GET" {
    savPA := gsh.gshPA
    var bsize int = 64*1024
    req = fmt.Sprintf("%v\r\n", strings.Join(argv, " "))
    fmt.Printf(Elapsed(Start)+"--In- C: %v", req)
    fmt.Fprintf(serv, req)
    count, err = serv.Read(res)
    if err != nil {
    }else{
        var dsize int64 = 0
        var out *os.File = nil
        var out_tobeclosed *os.File = nil
        var fname string = ""
        var rcode int = 0
        var pid int = -1
        fmt.Sscanf(string(res), "%d %d", &rcode, &dsize)
        fmt.Printf(Elapsed(Start)+"--In- S: %v", string(res[0:count]))
        if 3 <= len(argv) {
            fname = argv[2]
            if strBegins(fname, "{") {
                xin, xout, err := gsh.Popen(fname, "w")
                if err {
                }else{
                    xin.Close()
                    defer xout.Close()
                    out = xout
                    out_tobeclosed = xout
                    pid = 0 // should be its pid
                }
            }else{
                // should write to temporary file
                // should suppress ^C on tty
                xout, err := os.OpenFile(fname, os.O_CREATE|os.O_RDWR|os.O_TRUNC, 0600)
                if err != nil {
                    fmt.Print("--En- %v\n", err)
                }
                out = xout
            }
        }
        in, _ := serv.File()
        fileRelay("RecvGET", in, out, dsize, bsize)
        if 0 <= pid {
            gsh.gshPA = savPA // recovery of Fd(), and more?
            fmt.Printf(Elapsed(Start)+"--In- L: close Pipe > %v\n", fname)
            out_tobeclosed.Close()
            //syscall.Wait4(pid, nil, 0, nil) //@@
        }
    }
}
}
else
if argv[0] == "PUT" {
    remote, _ := serv.File()
    var local *os.File = nil
    var dsize int64 = 32*1024*1024
    var bsize int = 64*1024
    var ofile string = "-"
    //fmt.Printf("--I-- Rex %v\n", argv)
    if 1 < len(argv) {
        fname := argv[1]
        if strBegins(fname, "-z") {
            fmt.Sscanf(fname[2:], "%d", &dsize)
        }else
        if strBegins(fname, "{") {
            xin, xout, err := gsh.Popen(fname, "r")
            if err {
            }else{
                xout.Close()
                defer xin.Close()
                //in = xin
                local = xin
                fmt.Printf("--In- [%d] < Upload output of %v\n",
                    local.Fd(), fname)
                ofile = "-from."+fname
                dsize = MaxStreamSize
            }
        }else{
            xlocal, err := os.Open(fname)
            if err != nil {
                fmt.Printf("--En- (%s)\n", err)
                local = nil
            }else{
                local = xlocal
                fi, _ := local.Stat()
                dsize = fi.Size()
                defer local.Close()
                //fmt.Printf("--I-- Rex in(%v / %v)\n", ofile, dsize)
            }
            ofile = fname
            fmt.Printf(Elapsed(Start)+"--In- L: open(%v,r)=%v %v (%v)\n",
                fname, dsize, local, err)
        }
    }
}
if 2 < len(argv) && argv[2] != "" {
    ofile = argv[2]
    //fmt.Printf("(%d)%v B.ofile=%v\n", len(argv), argv, ofile)
}
//fmt.Printf(Elapsed(Start)+"--I-- Rex out(%v)\n", ofile)
fmt.Printf(Elapsed(Start)+"--In- PUT %v (%v)\n", dsize, bsize)
req = fmt.Sprintf("PUT %v %v \r\n", dsize, ofile)

```

```

        if debug { fmt.Printf(Elapsed(Start)+"--In- C: %v",req) }
        fmt.Fprintf(serv,"%v",req)
        count,err = serv.Read(res)
        if debug { fmt.Printf(Elapsed(Start)+"--In- S: %v",string(res[0:count])) }
        fileRelay("SendPUT",local,remote,dsize,bsize)
    }else{
        req = fmt.Sprintf("%v\r\n",strings.Join(argv," "))
        if debug { fmt.Printf(Elapsed(Start)+"--In- C: %v",req) }
        fmt.Fprintf(serv,"%v",req)
        //fmt.Printf("--In- sending RexRequest(%v)\n",len(req))
    }
    //fmt.Printf(Elapsed(Start)+"--In- waiting RexResponse...\n")
    count,err = serv.Read(res)
    res := ""
    if count == 0 {
        res = "(nil)\r\n"
    }else{
        res = string(res[:count])
    }
    if err != nil {
        fmt.Printf(Elapsed(Start)+"--En- S: (%d,%v) %v",count,err,res)
    }else{
        fmt.Printf(Elapsed(Start)+"--In- S: %v",res)
    }
    serv.Close()
    //conn.Close()
}

```

Remote Shell

```

// gcp file [...] { [host]:[port]:[dir] | dir } // -p | -no-p
func (gsh*GshContext)FileCopy(argv[]string){
    var host = ""
    var port = ""
    var upload = false
    var download = false
    var xargv = []string{"rex-gcp"}
    var srcv = []string{}
    var dstv = []string{}
    argv = argv[1:]

    for _,v := range argv {
        /*
        if v[0] == '-' { // might be a pseudo file (generated date)
            continue
        }
        */
        obj := strings.Split(v,":")
        //fmt.Printf("%d %v %v\n",len(obj),v,obj)
        if 1 < len(obj) {
            host = obj[0]
            file := ""
            if 0 < len(host) {
                gsh.LastServer.host = host
            }else{
                host = gsh.LastServer.host
                port = gsh.LastServer.port
            }
            if 2 < len(obj) {
                port = obj[1]
                if 0 < len(port) {
                    gsh.LastServer.port = port
                }else{
                    port = gsh.LastServer.port
                }
            }
            file = obj[2]
        }else{
            file = obj[1]
        }
        if len(srcv) == 0 {
            download = true
            srcv = append(srcv,file)
            continue
        }
        upload = true
        dstv = append(dstv,file)
        continue
    }
    /*
    idx := strings.Index(v,":")
    if 0 <= idx {
        remote = v[0:idx]
        if len(srcv) == 0 {
            download = true
            srcv = append(srcv,v[idx+1:])
            continue
        }
        upload = true
        dstv = append(dstv,v[idx+1:])
        continue
    }
    */
    if download {
        dstv = append(dstv,v)
    }else{
        srcv = append(srcv,v)
    }
}
hostport := "@" + host + ":" + port
if upload {
    if host != "" { xargv = append(xargv,hostport) }
    xargv = append(xargv,"PUT")
    xargv = append(xargv,srcv[0:]...)
    xargv = append(xargv,dstv[0:]...)
    //fmt.Printf("--I-- FileCopy PUT gsh://%s/%v < %v // %v\n",hostport,dstv,srcv,xargv)
    fmt.Printf("--I-- FileCopy PUT gsh://%s/%v < %v\n",hostport,dstv,srcv)
    gsh.RexecClient(xargv)
}else{
    if download {
        if host != "" { xargv = append(xargv,hostport) }
        xargv = append(xargv,"GET")
        xargv = append(xargv,srcv[0:]...)
        xargv = append(xargv,dstv[0:]...)
        //fmt.Printf("--I-- FileCopy GET gsh://%v/%v > %v // %v\n",hostport,srcv,dstv,xargv)
        fmt.Printf("--I-- FileCopy GET gsh://%v/%v > %v\n",hostport,srcv,dstv)
        gsh.RexecClient(xargv)
    }else{
    }
}
}

```

network

```

// -s, -si, -so // bi-directional, source, sync (maybe socket)
func sconnect(gshCtx GshContext, inTCP bool, argv []string) {
    gshPA := gshCtx.gshPA
    if len(argv) < 2 {

```

```

        fmt.Printf("Usage: -s [host]:[port[.udp]]\n")
        return
    }
    remote := argv[1]
    if remote == ":" { remote = "0.0.0.0:9999" }

    if inTCP { // TCP
        dport, err := net.ResolveTCPAddr("tcp", remote);
        if err != nil {
            fmt.Printf("Address error: %s (%s)\n", remote, err)
            return
        }
        conn, err := net.DialTCP("tcp", nil, dport)
        if err != nil {
            fmt.Printf("Connection error: %s (%s)\n", remote, err)
            return
        }
        file, _ := conn.File();
        fd := file.Fd()
        fmt.Printf("Socket: connected to %s, socket[%d]\n", remote, fd)

        savfd := gshPA.Files[1]
        gshPA.Files[1] = fd;
        gshelly(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;
        gshelly(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;
        gshelly(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)
        gshelly(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):
        gshCtx.ShowChdirHistory(argv)
    case isin("-ls",argv):
        showFileInfo(cwd,argv)
    default:
        fmt.Printf("%s\n",cwd)
    case isin("-v",argv): // obsolete empty 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)
        fmt.Printf("\n")
    }
}

// History
// these should be browsed and edited by HTTP browser
// show the time of command with -t and direcotry with -ls
// openfile-history, sort by -a -m -c
// sort by elapsed time by -t -s
// search by "more" like interface
// edit history
// sort history, and wc or uniq
// CPU and other resource consumptions
// limit showing range (by time or so)
// export / import history
func xHistory(gshCtx GshContext, argv []string) (rgshCtx GshContext) {
    atWorkDirX := -1
    if 1 < len(argv) && strBegins(argv[1],"@") {
        atWorkDirX,_ = strconv.Atoi(argv[1][1:])
    }
    //fmt.Printf("--D-- showHistory(%v)\n",argv)
    for i, v := range gshCtx.CommandHistory {
        // exclude commands not to be listed by default
        // internal commands may be suppressed by default
        if v.CmdLine == "" && !isin("-a",argv) {
            continue;
        }
        if 0 <= atWorkDirX {
            if v.WorkDirX != atWorkDirX {
                continue
            }
        }
        if !isin("-n",argv){ // like "fc"
            fmt.Printf("!%2d ",i)
        }
        if isin("-v",argv){
            fmt.Println(v) // should be with it date
        }else{
            if isin("-l",argv) || isin("-l0",argv) {
                elps := v.EndAt.Sub(v.StartAt);
                start := v.StartAt.Format(time.Stamp)
                fmt.Printf("@%d ",v.WorkDirX)
                fmt.Printf("[%v] %11v/t ",start,elps)
            }
            if isin("-l",argv) && !isin("-l0",argv){
                fmt.Printf("%v",Rusagef("%t %u\t// %s",argv,v.Rusagev))
            }
            if isin("-at",argv) { // isin("-ls",argv){
                dhi := v.WorkDirX // workdir history index
                fmt.Printf("@%d %s\t",dhi,v.WorkDirX)
                // show the FileInfo of the output command??
            }
            fmt.Printf("%s",v.CmdLine)
            fmt.Printf("\n")
        }
    }
    return gshCtx
}

// !n - history index
func searchHistory(gshCtx GshContext, gline string) (string, bool, bool){
    if gline[0] == '!' {
        hix, err := strconv.Atoi(gline[1:])
        if err != nil {
            fmt.Printf("--E-- (%s : range)\n",hix)
            return "", false, true
        }
        if hix < 0 || len(gshCtx.CommandHistory) <= hix {
            fmt.Printf("--E-- (%d : out of range)\n",hix)
            return "", false, true
        }
        return gshCtx.CommandHistory[hix].CmdLine, false, false
    }
    // search
    //for i, v := range gshCtx.CommandHistory {
    //}
    return gline, false, false
}

// temporary adding to PATH environment
// cd name -lib for LD_LIBRARY_PATH
// chdir with directory history (date + full-path)
// -s for sort option (by visit date or so)
func (gsh*GshContext)ShowChdirHistory1(i int,v GChdirHistory, argv []string){
    fmt.Printf("!%2d ",v.CmdIndex) // the first command at this WorkDir
    fmt.Printf("@%d ",i)
    fmt.Printf("[%v]",v.MovedAt.Format(time.Stamp))
    showFileInfo(v.Dir,argv)
}

func (gsh*GshContext)ShowChdirHistory(argv []string){
    for i, v := range gsh.ChdirHistory {
        gsh.ShowChdirHistory1(i,v,argv)
    }
}

func skipOpts(argv []string)(int){
    for i,v := range argv {

```



```

        if strBegins(v, "-") {
        }else{
            return i
        }
    }
    return -1
}
func xChdir(gshCtx GshContext, argv []string) (rgshCtx GshContext) {
    cdhist := gshCtx.CkdirHistory
    if isin("?", argv) || isin("-", argv) || isin("-a", argv) {
        gshCtx.ShowCkdirHistory(argv)
        return gshCtx
    }
    pwd, _ := os.Getwd()
    dir := ""
    if len(argv) <= 1 {
        dir = toFullpath("-")
    }else{
        i := skipOpts(argv[1:])
        if i < 0 {
            dir = toFullpath("-")
        }else{
            dir = argv[1+i]
        }
    }
    if strBegins(dir, "@") {
        if dir == "@0" { // obsolete
            dir = gshCtx.StartDir
        }else{
            if dir == "@1" {
                index := len(cdhist) - 1
                if 0 < index { index -= 1 }
                dir = cdhist[index].Dir
            }else{
                index, err := strconv.Atoi(dir[1:])
                if err != nil {
                    fmt.Printf("--E-- xChdir(%v)\n", err)
                    dir = "?"
                }else{
                    if len(gshCtx.CkdirHistory) <= index {
                        fmt.Printf("--E-- xChdir(history range error)\n")
                        dir = "?"
                    }else{
                        dir = cdhist[index].Dir
                    }
                }
            }
        }
    }
    if dir != "?" {
        err := os.Chdir(dir)
        if err != nil {
            fmt.Printf("--E-- xChdir(%s)(%v)\n", argv[1], err)
        }else{
            cwd, _ := os.Getwd()
            if cwd != pwd {
                hist1 := GCkdirHistory { }
                hist1.Dir = cwd
                hist1.MovedAt = time.Now()
                hist1.CmdIndex = len(gshCtx.CommandHistory)+1
                gshCtx.CkdirHistory = append(cdhist, hist1)
                if !isin("-s", argv){
                    //cwd, _ := os.Getwd()
                    //fmt.Printf("%s\n", cwd)
                    ix := len(gshCtx.CkdirHistory)-1
                    gshCtx.ShowCkdirHistory1(ix, hist1, argv)
                }
            }
        }
    }
    if isin("-ls", argv){
        cwd, _ := os.Getwd()
        showFileInfo(cwd, argv);
    }
    return gshCtx
}
func TimeValSub(tv1 *syscall.Timeval, tv2 *syscall.Timeval){
    *tv1 = syscall.NsecToTimeval(tv1.Nano() - tv2.Nano())
}
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
}
*/
// Resource Usage
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("Uusr=%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)
}

```



```

        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
}
}

// Plugin
// plugin [-ls [names]] to list plugins
// Reference: plugin 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")
    }
}
}

// Scanf // string decomposer
// scanf [format] [input]
func scanf(sstr string)(strv[]string){
    strv = strings.Split(sstr," ")
    return strv
}
func scanUntil(src,end string)(rstr string,leng int){
    idx := strings.Index(src,end)
    if 0 <= idx {
        rstr = src[0:idx]
        return rstr,idx+leng(end)
    }
    return src,0
}
}

// -bn -- display base-name part only // can be in some %fmt, for sed rewriting
func (gsh*GshContext)printVal(fmts string, vstr string, optv[]string){
    //vint,err := strconv.Atoi(vstr)
    var ival int64 = 0
    n := 0
    err := error(nil)
    if strBegins(vstr, "-") {
        vx, _ := strconv.Atoi(vstr[1:])
        if vx < len(gsh.iValues) {
            vstr = gsh.iValues[vx]
        }else{
        }
    }
    // should use Eval()
    if strBegins(vstr,"0x") {
        n,err = fmt.Sscanf(vstr[2:],"%x",&ival)
    }else{
        n,err = fmt.Sscanf(vstr,"%d",&ival)
    }
    //fmt.Printf("--D-- n=%d err=(%v) {%s}=%v\n",n,err,vstr, ival)
}
if n == 1 && err == nil {
    //fmt.Printf("--D-- formatn(%v) ival(%v)\n",fmts,ival)
}
}

```

```

        fmt.Printf("%"+fmts,ival)
    }else{
        if isin("-bn",optv){
            fmt.Printf("%"+fmts,filepath.Base(vstr))
        }else{
            fmt.Printf("%"+fmts,vstr)
        }
    }
}
func (gsh*GshContext)printfv(fmts,div string,argv[]string,optv[]string,list[]string){
    //fmt.Printf("%d",len(list))
    //curfmt := "v"
    outlen := 0
    curfmt := gsh.iFormat

    if 0 < len(fmts) {
        for xi := 0; xi < len(fmts); xi++ {
            fch := fmts[xi]
            if fch == '%' {
                if xi+1 < len(fmts) {
                    curfmt = string(fmts[xi+1])
                    gsh.iFormat = curfmt
                    xi += 1
                }
                if xi+1 < len(fmts) && fmts[xi+1] == '(' {
                    vals, leng := scanUntil(fmts[xi+2:],")")
                    //fmt.Printf("--D-- show fmt(%v) val(%v) next(%v)\n",curfmt,vals,leng)
                    gsh.printVal(curfmt,vals,optv)
                    xi += 2+leng-1
                    outlen += 1
                }
                continue
            }
            if fch == '.' {
                hi,leng := scanInt(fmts[xi+1:])
                if 0 < leng {
                    if hi < len(gsh.iValues) {
                        gsh.printVal(curfmt,gsh.iValues[hi],optv)
                        outlen += 1 // should be the real length
                    }else{
                        fmt.Printf("(out-range)")
                    }
                    xi += leng
                    continue;
                }
            }
            fmt.Printf("%c",fch)
            outlen += 1
        }
    }
    }else{
        //fmt.Printf("--D-- print {%s}\n")
        for i,v := range list {
            if 0 < i {
                fmt.Printf(div)
            }
            gsh.printVal(curfmt,v,optv)
            outlen += 1
        }
    }
    if 0 < outlen {
        fmt.Printf("\n")
    }
}
func (gsh*GshContext)Scanv(argv[]string){
    //fmt.Printf("--D-- Scnav(%v)\n",argv)
    if len(argv) == 1 {
        return
    }
    argv = argv[1:]
    fmts := ""
    if strBegins(argv[0],"-F") {
        fmts = argv[0]
        gsh.iDelimiter = fmts
        argv = argv[1:]
    }
    input := strings.Join(argv," ")
    if fmts == "" { // simple decomposition
        v := scanv(input)
        gsh.iValues = v
        //fmt.Printf("%v\n",strings.Join(v," "))
    }else{
        v := make([]string,8)
        n,err := fmt.Sscanf(input,fmts,&v[0],&v[1],&v[2],&v[3])
        fmt.Printf("--D-- Sscanf ->(%v) n=%d err=(%v)\n",v,n,err)
        gsh.iValues = v
    }
}
func (gsh*GshContext)Printv(argv[]string){
    if false { //000
        fmt.Printf("%v\n",strings.Join(argv[1:], " "))
        return
    }
    //fmt.Printf("--D-- Printv(%v)\n",argv)
    //fmt.Printf("%v\n",strings.Join(gsh.iValues," "))
    div := gsh.iDelimiter
    fmts := ""
    argv = argv[1:]
    if 0 < len(argv) {
        if strBegins(argv[0],"-F") {
            div = argv[0][2:]
            argv = argv[1:]
        }
    }

    optv := []string{}
    for _,v := range argv {
        if strBegins(v,"-"){
            optv = append(optv,v)
            argv = argv[1:]
        }else{
            break;
        }
    }
    if 0 < len(argv) {
        fmts = strings.Join(argv," ")
    }
    gsh.printfv(fmts,div,argv,optv,gsh.iValues)
}
func (gsh*GshContext)Basename(argv[]string){
    for i,v := range gsh.iValues {
        gsh.iValues[i] = filepath.Base(v)
    }
}
func (gsh*GshContext)Sortv(argv[]string){
    sv := gsh.iValues
}

```

```

    sort.Slice(sv , func(i,j int) bool {
        return sv[i] < sv[j]
    })
}
func (gsh*GshContext)Shiftv(argv[]string){
    vi := len(gsh.iValues)
    if 0 < vi {
        if isin("-r",argv) {
            top := gsh.iValues[0]
            gsh.iValues = append(gsh.iValues[1:],top)
        }else{
            gsh.iValues = gsh.iValues[1:]
        }
    }
}
func (gsh*GshContext)Enq(argv[]string){
}
func (gsh*GshContext)Deq(argv[]string){
}
func (gsh*GshContext)Push(argv[]string){
    gsh.iValStack = append(gsh.iValStack,argv[1:])
    fmt.Printf("depth=%d\n",len(gsh.iValStack))
}
func (gsh*GshContext)Dump(argv[]string){
    for i,v := range gsh.iValStack {
        fmt.Printf("%d %v\n",i,v)
    }
}
func (gsh*GshContext)Pop(argv[]string){
    depth := len(gsh.iValStack)
    if 0 < depth {
        v := gsh.iValStack[depth-1]
        if isin("-cat",argv){
            gsh.iValues = append(gsh.iValues,v...)
        }else{
            gsh.iValues = v
        }
        gsh.iValStack = gsh.iValStack[0:depth-1]
        fmt.Printf("depth=%d %s\n",len(gsh.iValStack),gsh.iValues)
    }else{
        fmt.Printf("depth=%d\n",depth)
    }
}
}

// Command Interpreter
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,strings.TrimSpace(argv[ai]))
    }
    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{}); // empty command
    case cmd == "-x":
        gshCtx.CmdTrace = !gshCtx.CmdTrace
    case cmd == "-xt":
        gshCtx.CmdTime = !gshCtx.CmdTime
    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 == "-bn":
        gshCtx.Basename(argv)
    case cmd == "call":
        _,_ = gshCtx.excommand(false,argv[1:])
    case cmd == "cd" || cmd == "chdir":
        gshCtx = xChdir(gshCtx,argv);
    case cmd == "close":
        gshCtx = xClose(gshCtx,argv)
    case cmd == "gcp":
        gshCtx.FileCopy(argv)
    case cmd == "dec" || cmd == "decode":
        Dec(&gshCtx,argv)
    case cmd == "#define":
    case cmd == "dump":
        gshCtx.Dump(argv)
    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(true,argv[1:])
        // should not return here
    case cmd == "exit" || cmd == "quit":
        // write Result code EXIT to 3>
        return gshCtx, true
    case cmd == "fds":
        // dump the attributes of fds (of other process)
    case cmd == "-find" || cmd == "fin" || cmd == "ufind" || cmd == "uf":
        gshCtx.xFind(argv[1:])
    case cmd == "fu":

```

```

        gshCtx.xFind(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(argv)
    case cmd == "gdeg":
        gshCtx.Beg(argv)
    case cmd == "genq":
        gshCtx.Eng(argv)
    case cmd == "gpop":
        gshCtx.Pop(argv)
    case cmd == "gpush":
        gshCtx.Push(argv)
    case cmd == "history" || cmd == "hi": // hi should be alias
        gshCtx = xHistory(gshCtx, argv)
    case cmd == "jobs":
        xJobs(gshCtx,argv)
    case cmd == "lnsp":
        SplitLine(&gshCtx,argv)
    case cmd == "-ls":
        gshCtx.xFind(argv)
    case cmd == "nop":
        // do nothing
    case cmd == "pipe":
        gshCtx = xOpen(gshCtx,argv)
    case cmd == "plug" || cmd == "plugin" || cmd == "pin":
        gshCtx._ = xPlugin(gshCtx,argv[1:])
    case cmd == "print" || cmd == "-pr":
        // output internal slice // also sprintf should be
        gshCtx.Printv(argv)
    case cmd == "ps":
        xPs(gshCtx,argv)
    case cmd == "pstitle":
        // to be gsh.title
    case cmd == "rexecd" || cmd == "rexed":
        gshCtx.RexecServer(argv)
    case cmd == "rexec" || cmd == "rex":
        gshCtx.RexecClient(argv)
    case cmd == "repeat" || cmd == "rep": // repeat cond command
        repeat(gshCtx,argv)
    case cmd == "scan":
        // scan input (or so in fscanf) to internal slice (like Files or map)
        gshCtx.Scanv(argv)
    case cmd == "set":
        // set name ...
    case cmd == "serv":
        httpServer(gshCtx,argv)
    case cmd == "shift":
        gshCtx.Shiftv(argv)
    case cmd == "sleep":
        sleep(gshCtx,argv)
    case cmd == "-sort":
        gshCtx.Sortv(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{
            notfound,_ := gshCtx.excommand(false,argv)
            if notfound {
                fmt.Printf("--E-- command not found (%v)\n",cmd)
            }
        }
    }
    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);
    if gshCtx.CmdTime {
        fmt.Printf("--T-- " + time.Now().Format(time.Stamp) + " (%d.%09ds)\n",
            elps/1000000000,elps%100000000)
    }
    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
}
// Command Line Editor
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()
        gline := string(buff[0:count])
        return gline
    }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(),0} }, // ChdirHistory
        gshPA,
        []GCommandHistory{}, //something for invocation?
        GCommandHistory{}, // CmdCurrent
        false,
        []int{},
        syscall.Rusage{},
        "", // GshHomeDir
        Ttyid(),
        false,
        false,
        []PluginInfo{},
        []string{},
        "",
        "v",
        ValueStack{},
        GServer{"",""}, // LastServer
    }
    err := false
    gshCtx, err = gshSetupHomedir(gshCtx)
    return gshCtx, err
}
// Main loop
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 :=
    gsh_getlinev, 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 {
                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
        }
        if false {
            os.Stdout.Write([]byte("gotline:"))
            os.Stdout.Write([]byte(gline))
            os.Stdout.Write([]byte("\n"))
        }
        gline = strsubst(&gshCtx,gline,true)
        if false {
            fmt.Printf("fmt.Printf %v - %v\n",gline)
            fmt.Printf("fmt.Printf %s - %s\n",gline)
            fmt.Printf("fmt.Printf %x - %s\n",gline)
            fmt.Printf("fmt.Printf %U - %s\n",gline)
            fmt.Printf("Stoutt.Write -")
            os.Stdout.Write([]byte(gline))
            fmt.Printf("\n")
        }
        /*
        // 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.WorkDirX = len(gshCtx.ChdirHistory)-1
        //fmt.Printf("--D--ChdirHistory(%d)\n",len(gshCtx.ChdirHistory))
        ghist.StartAt = time.Now()
        rusagev1 := Getrusagev()
        gshCtx.CmdCurrent.FoundFile = []string{}
        xgshCtx, fin := tgshell1(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)
    //gshell1(gshCtx,"time")
}
//

```

▼ Consideration

```

// - 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 subcommand 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 quotation 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
// - filepath substitution shold be done by each command, expecially in case of builtins
// - @N substitution for the history of working directory, and @spec for more generic ones
// - @dir prefix to do the command at there, that means like (chdir @dir; command)
// - GSH_PATH for plugins
// - standard command output: list of data with name, size, resouce usage, modified time
// - generic sort key option -nm name, -sz size, -ru rusage, -ts start-time, -tm mod-time
// - -wc word-count, grep match line count, ...
// - standard command execution result: a list of string, -tm, -ts, -ru, -sz, ...
// - -tailf-filename like tail -f filename, repeat close and open before read
// - max. size and max. duration and timeout of (generated) data transfer
//---END--- (^-^)/ITS more

```

▶ References



→ 1/1