

// *

GShell version 0.1.4 // 2020-08-18 // SatoxITS

hell ==GShell ==GShell ==GShell ==GShell ==GShell

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
[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 = "0.1.4"
var DATE = "2020-0818a"
var LINESIZE = (8*1024)
var PATHSEP = ";" // should be ":" in Windows
var DIRSEP = "/" // canbe \ in Windows
var GSH_HOME = ".gsh" // under home directory
var PROMPT = ">"

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

// 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
    Rusage     [2]syscall.Rusage // Resource consumption, CPU time or so
    CmdId     int       // maybe with identified with arguments or impact
    WorkDir   string   // redirection commands should not be the CmdId
    CmdLine   string   // working directory at start
    CmdLine   string   // command line
}

type GChdirHistory struct {
    Dir        string
    MovedAt   time.Time
}

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 GshContext struct {
    StartDir string // the current directory at the start
    GetLine  string // gsh-getline command as a input line editor
    ChdirHistory []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)
}

```

```

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,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
}

```

```

// str-expansion
// - 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 == '@':
                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 %v %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("\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*,lab,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 xGrep(gshCtx GshContext, path string, rexp []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, rexp) }
    //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), rexp[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
    MaxNmlen int64 // max. name length
    nextRepo time.Time
}
func showFusage(dir string, fusage *fileSum) {
    bsum := 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, bsum);
}
const (
    S_IFMT = 0170000
    S_IFCHR = 0020000
    S_IFDIR = 0040000

```

```

    S_IFREG   = 0100000
    S_IFLNK   = 0120000
    S_IFSOCK  = 0140000
)
func cumPinfo(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)
            showPusage(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("%8dBk %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].CreateTime().Sub(filev[j].CreateTime())
        })
    }
    */
    /*
    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 = 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 := 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)
                }
            }
        }
    }
}

```

```

    }
}
return gshCtx,total
}
func xxFind(gshCtx GshContext,depth int,total *fileSum,dir string,npatv[ ]string)(GshContext,*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
stater := syscall.Lstat(dir,&dstat) // should be flstat
if stater != nil {
if !isin("-w",argv){ fmt.Printf("ufind: %v\n",stater) }
return gshCtx,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,stater,dstat,argv,true)
if !nols && !isin("-s",argv) && (!isin("-du",argv) || isin("-a",argv)) {
showFileInfo(dir,argv)
}
}
// 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)
}
gshCtx,total = xxFindEntv(gshCtx,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 gshCtx,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 xFind(gshCtx GshContext,argv[ ]string)(GshContext){
if 0 < len(argv) && strBegins(argv[0],"?"){
showFound(gshCtx,argv)
return gshCtx
}
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{"*"} }
gshCtx,fusage := xxFind(gshCtx,0,&total,cwd,npats,argv)
if !isin("-grep",argv) {
showFusage("total",fusage)
}
return gshCtx
}
}
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(" ") //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 executable?
        _, 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")
    }
}
// External commands
func excommand(gshCtx GshContext, exec bool, argv []string) (GshContext, bool) {
    if gshCtx.CmdTrace { fmt.Printf("--I-- excommand[%v] [%v]\n", exec, argv) }
}

```

```

gshPA := gshCtx.gshPA
fullpathv, itis := which("PATH",[]string{"which",argv[0],"-s"})
if itis == false {
    return gshCtx, true
}
fullpath := fullpathv[0]
argv = unescapeWhiteSPV(argv)
if 0 < strings.Index(fullpath, ".go") {
    nargv := argv // []string{}
    gofullpathv, itis := which("PATH",[]string{"which", "go", "-s"})
    if itis == false {
        fmt.Printf("--F-- Go not found\n")
        return gshCtx, true
    }
    gofullpath := gofullpathv[0]
    nargv = []string{ gofullpath, "run", fullpath }
    fmt.Printf("--I-- %s (%s %s %s)\n", gofullpath,
        nargv[0], nargv[1], nargv[2])
    if exec {
        syscall.Exec(gofullpath, nargv, os.Environ())
    } else {
        pid, _ := syscall.ForkExec(gofullpath, nargv, &gshPA)
        if gshCtx.Background {
            fmt.Printf("--I-- in Background [%d]\n", pid)
            gshCtx.BackgroundJobs = append(gshCtx.BackgroundJobs, pid)
        } else {
            rusage := syscall.Rusage {}
            syscall.Wait4(pid, nil, 0, &rusage)
            gshCtx.LastRusage = rusage
            gshCtx.CmdCurrent.Rusagev[1] = rusage
        }
    }
} else {
    if exec {
        syscall.Exec(fullpath, argv, os.Environ())
    } else {
        pid, _ := syscall.ForkExec(fullpath, argv, &gshPA)
        //fmt.Printf("[%d]\n", pid); // 's' to be background
        if gshCtx.Background {
            fmt.Printf("--I-- in Background [%d]\n", pid)
            gshCtx.BackgroundJobs = append(gshCtx.BackgroundJobs, pid)
        } else {
            rusage := syscall.Rusage {}
            syscall.Wait4(pid, nil, 0, &rusage);
            gshCtx.LastRusage = rusage
            gshCtx.CmdCurrent.Rusagev[1] = rusage
        }
    }
}
return gshCtx, 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()
}

// 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;
    gshellv(gshCtx, argv[2:])
    gshPA.Files[1] = savfd
    file.Close()
    conn.Close()
}
}

//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();
    }
    //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 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) {
    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 !isin("-n",argv){ // like "fc"
            fmt.Printf("!%3d ",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("%s %11v/t ",start,elps)
            }
            if isin("-l",argv) && !isin("-l0",argv){
                fmt.Printf("%v",Rusagef("%t %u %s",argv,v.Rusagev))
            }
            if isin("-ls",argv){
                fmt.Printf("@%s ",v.WorkDir)
                // show the FileInfo of the output command??
            }
            fmt.Printf("%s",v.CmdLine)
            fmt.Printf("\n")
        }
    }
    return gshCtx
}

// In - 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 xChdirHistory(gshCtx GshContext, argv []string){
    for i, v := range gshCtx.CkdirHistory {
        fmt.Printf("!%d ",i)
        fmt.Printf("%v ",v.MovedAt.Format(time.Stamp))
        showFileInfo(v.Dir,argv)
    }
}

func xChdir(gshCtx GshContext, argv []string) (rgshCtx GshContext) {
    cdhist := gshCtx.CkdirHistory
    if isin("?",argv) || isin("-t",argv) {
        xChdirHistory(gshCtx,argv)
        return gshCtx
    }
    pwd, _ := os.Getwd()
    dir := ""
    if len(argv) <= 1 {
        dir = toFullpath("-")
    }else{
        dir = argv[1]
    }
    if strBegins(dir,"!") {
        if dir == "!0" {
            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 := GChdirHistory { }
                hist1.Dir = cwd
                hist1.MovedAt = time.Now()
                gshCtx.CkdirHistory = append(cdhist,hist1)
            }
        }
    }
    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("Utr=%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 {}
        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" command
func xOpen(gshCtx GshContext, argv []string)(GshContext){
    var pv = [2]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
}
}

// redirect
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
}

// 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)
    }
}

```



```

}
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:]
        }
    }
}

// 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, 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{}); // 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 == "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 == "fdls":
        // dump the attributes of fds (of other process)
    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 == "lnsp":
        SplitLine(&gshCtx,argv)
    case cmd == "-ls":
        gshCtx = xFind(gshCtx,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":
        // 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 == "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{
            gshCtx, _ = excommand(gshCtx,false,argv)
        }
    }
    return gshCtx, fin
}

func gshell(gshCtx GshContext, gline string) (gx GshContext, rfin bool) {
    argv := strings.Split(string(gline), " ")
    gshCtx, fin := gshellv(gshCtx,argv)
    return gshCtx, fin
}

func tgshell(gshCtx GshContext, gline string) (gx GshContext, xfin bool) {
    start := time.Now()
    gshCtx, fin := gshell(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%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
}

// Command Line Editor
func getline(gshCtx GshContext, hix int, skipping, with_exgetline bool, gsh_getline[]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_getline[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-ro.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 starting 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,
        false,
        []PluginInfo{},
        []string{},
        "",
        "v",
    }
    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_getline, with_exgetline :=
        which("PATH",[]string{"which","gsh-getline","-s"})
    if with_exgetline {
        gsh_getline[0] = toFullpath(gsh_getline[0])
        gshCtx.GetLine = toFullpath(gsh_getline[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_getline,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 := 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
//---END--- (~~/ITS more

```

► References

