

```
//
// gsh - Go lang based Shell
// (c) 2020 ITS more Co., Ltd.
// 2020-0807 created by SatoxITS (sato@its-more.jp)
//
package main // gsh main
// Documents: https://golang.org/pkg/
import (
    "bufio"
    "strings"
    "strconv"
    "sort"
    "fmt"
    "os"
    "time"
    "syscall"
    "go/types"
    "go/token"
    "net"
    "net/http" // http
    "text/html" // html
    "io/ioutil"
    "path/filepath" // for wildcard Match()
)

var VERSION = "gsh/0.0.6 (2020-0812a)"
var LINESIZE = (8*1024)
var PATHSEP = ":" // should be ";" in Windows
var DIRSEP = "/" // canbe \ in Windows
var PROMPT = "> "
var GSH_HOME = ".gsh" // under home directory

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

type GChdirHistory struct {
    Dir          string
    MovedAt      time.Time
}

type CmdMode struct {
    Background    bool
}

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
}

func isin(what string, list []string) bool {
    for _, v := range list {
        if v == what {
            return true
        }
    }
    return false
}

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)
    }
}

func strsubst(str string) string {
    rstr := ""
    inEsc := 0 // escape characer mode
    for _, ch := range str {
        if inEsc == 0 {
            if ch == '\\\ ' {

```

```

                inEsc = '\\\'
                continue
            }
            if ch == '%' {
                inEsc = '%'
                continue
            }
        }
        if inEsc == '\\\' {
            if ch == 's' { ch = ' ' }
            if ch == 'r' { ch = '\r' }
            if ch == 'n' { ch = '\n' }
            if ch == 't' { ch = '\t' }
            if ch == '\\\' { ch = '\\\' }
            inEsc = 0
        }
        if inEsc == '%' {
            if ch == '%' { ch = '%' }
            if ch == 'T' {
                rstr = rstr + time.Now().Format(time.Stamp)
                continue;
            }
            inEsc = 0
        }
        rstr = rstr + string(ch)
    }
    return rstr
}

func showFileInfo(path string, opts []string) {
    if 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("-n",opts) {
        fmt.Printf("\n")
    }
}

func toFullpath(path string) (fullpath string) {
    pathv := strings.Split(path,DIRSEP)
    if pathv[0] == "." {
        pathv[0], _ = os.Getwd()
    }else
    if pathv[0] == ".." {
        cwd, _ := os.Getwd()
        ppathv := strings.Split(cwd,DIRSEP)
        pathv[0] = strings.Join(ppathv,DIRSEP)
    }else
    if pathv[0] == "~" {
        pathv[0], _ = os.UserHomeDir()
    }
    return strings.Join(pathv,DIRSEP)
}

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){
    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() {
            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]
    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 find(argv []string){
}

func eval(argv []string, nlend bool){
    var ai = 1
    pfmt := "%s"
    if argv[ai][0:1] == "%" {
        pfmt = argv[ai]
        ai = 2
    }
    if len(argv) <= ai {
        return
    }
    gocode := strings.Join(argv[ai:], " ");
    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")
    }
}

func excommand(gshCtx GshContext, exec bool, argv []string) (GshContext, bool) {
    if gshCtx.CmdTrace { fmt.Printf("--I-- excommand[%d](%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]
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.Rusage = rusage
        }
    }
} else {
    if exec {
        syscall.Exec(fullpath, argv, os.Environ())
    } else {
        pid, _ := syscall.ForkExec(fullpath, argv, &gshPA)
        //fmt.Printf("[%d]\n", pid); // '&' 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.Rusage = rusage
        }
    }
}
return gshCtx, false
}
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 be 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()
}

// -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()
    } else {
        //dport, err := net.ResolveUDPAddr("udp4", remote);
        dport, err := net.ResolveUDPAddr("udp", remote);
        if err != nil {
            fmt.Printf("Address error: %s (%s)\n", remote, err)
            return
        }
        //conn, err := net.DialUDP("udp4", nil, dport)
        conn, err := net.DialUDP("udp", nil, dport)
        if err != nil {
            fmt.Printf("Connection error: %s (%s)\n", remote, err)
            return
        }
        file, _ := conn.File();
        fd := file.Fd()

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

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

func saccept(gshCtx GshContext, inTCP bool, argv []string) {
    gshPA := gshCtx.gshPA
    if len(argv) < 2 {
        fmt.Printf("Usage: -ac [host]:[port[.udp]]\n")
        return
    }
    local := argv[1]
    if local == ":" { local = "0.0.0.0:9999" }
    if inTCP { // TCP
        port, err := net.ResolveTCPAddr("tcp", local);
        if err != nil {
            fmt.Printf("Address error: %s (%s)\n", local, err)
            return
        }
        //fmt.Printf("Listen at %s...\n", local);
        sconn, err := net.ListenTCP("tcp", port)
        if err != nil {
            fmt.Printf("Listen error: %s (%s)\n", local, err)
            return
        }
    }
}

```

```

    }
    //fmt.Printf("Accepting at %s...\n",local);
    aconn, err := sconn.AcceptTCP()
    if err != nil {
        fmt.Printf("Accept error: %s (%s)\n",local,err)
        return
    }
    file, _ := aconn.File()
    fd := file.Fd()
    fmt.Printf("Accepted TCP at %s [%d]\n",local,fd)

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

    sconn.Close();
    aconn.Close();
    file.Close();
}
}

//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()

    if isin("-a",argv){
        // show directory history
    }
    if isin("-v",argv) {
        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")
    }
    else
    if isin("-ls",argv){
        showFileInfo(cwd,argv)
    }
    else{
        fmt.Printf("%s\n",cwd)
    }
}

}

// 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.Rusage))
            }
            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
}
// !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 xChdirHistory(gshCtx GshContext, argv []string){
    for i, v := range gshCtx.ChdirHistory {
        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.ChdirHistory
    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 dir[0] == '!' {
        if dir == "!0" {
            dir = gshCtx.StartDir
        }else
        if dir == "!" {
            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.ChdirHistory) <= 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.ChdirHistory = append(cdhist, hist1)
            }
        }
    }
    if isin("-ls", argv) {
        cwd, _ := os.Getwd()
        showFileInfo(cwd, argv);
    }
    return gshCtx
}
func RusageSub(rul, ru2 *syscall.Rusage) {
    rul.Utime.Sec -= ru2.Utime.Sec
    rul.Utime.Usec -= ru2.Utime.Usec
    rul.Stime.Sec -= ru2.Stime.Sec
    rul.Stime.Usec -= ru2.Stime.Usec
}
func Rusagef(fmtspec string, argv []string, ru *syscall.Rusage) (string) {
    fmt.Printf("%d.%06ds/u ", ru.Utime.Sec, ru.Utime.Usec)
    fmt.Printf("%d.%06ds/s ", ru.Stime.Sec, ru.Stime.Usec)
    return ""
}
func showRusage(what string, argv []string, ru *syscall.Rusage) {
    fmt.Printf("%s: ", what);
    fmt.Printf("Usr=%d.%06ds", ru.Utime.Sec, ru.Utime.Usec)
    fmt.Printf(" Sys=%d.%06ds", ru.Stime.Sec, ru.Stime.Usec)
    fmt.Printf(" Rss=%vB", ru.Maxrss)
    if isin("-l", argv) {
        fmt.Printf(" MinFlt=%v", ru.Minflt)
        fmt.Printf(" MajFlt=%v", ru.Majflt)
        fmt.Printf(" IxRSS=%vB", ru.Ixrss)
        fmt.Printf(" IdRSS=%vB", ru.Idrss)
        fmt.Printf(" Nswap=%vB", ru.Nswap)
        fmt.Printf(" Read=%v", ru.Inblock)
        fmt.Printf(" Write=%v", ru.Oublock)
    }
    fmt.Printf(" Snd=%v", ru.Msgsnd)
    fmt.Printf(" Rcv=%v", ru.Msgrcv)
    //if isin("-l", argv) {
        fmt.Printf(" Sig=%v", ru.Nsignals)
    //}
    fmt.Printf("\n");
}
func xTime(gshCtx GshContext, argv []string) (GshContext, bool) {
    if 2 <= len(argv) {
        xgshCtx, fin := gshellv(gshCtx, argv[1:])
        gshCtx = xgshCtx
        showRusage(argv[1], argv, &gshCtx.LastRusage)
        return gshCtx, fin
    } else {
        rusage := syscall.Rusage { }
        syscall.Getrusage(syscall.RUSAGE_SELF, &rusage)
        showRusage("self", argv, &rusage)
        syscall.Getrusage(syscall.RUSAGE_CHILDREN, &rusage)
        showRusage("chld", argv, &rusage)
        return gshCtx, false
    }
}
func xJobs(gshCtx GshContext, argv []string) {
    fmt.Printf("%d Jobs\n", len(gshCtx.BackGroundJobs))
    for ji, pid := range gshCtx.BackGroundJobs {
        //wstat := syscall.WaitStatus {0}
        rusage := syscall.Rusage { }
        //wpid, err := syscall.Wait4(pid, &wstat, syscall.WNOHANG, &rusage);
        wpid, err := syscall.Wait4(pid, nil, syscall.WNOHANG, &rusage);
        if err != nil {
            fmt.Printf("--E-- %%d [%d] (%v)\n", ji, pid, err)
        } else {
            fmt.Printf("%%d [%d] (%d)\n", ji, pid, wpid)
            showRusage("chld", argv, &rusage)
        }
    }
}
func inBackground(gshCtx GshContext, argv []string) (GshContext, bool) {
    if gshCtx.CmdTrace { fmt.Printf("--I-- inBackground(%v)\n", argv) }
    gshCtx.BackGround = true // set background option
    xfin := false
    gshCtx, xfin = gshellv(gshCtx, argv)
    gshCtx.BackGround = false
    return gshCtx, xfin
}

```



```

// -o file without command means just opening it and refer by #N
// should be listed by "files" command
func xOpen(gshCtx GshContext, argv []string) (GshContext) {
    var pv = []int{-1,-1}
    err := syscall.Pipe(pv)
    fmt.Printf("--I-- pipe(%d,%d)=%v\n",pv[0],pv[1],err)
    return gshCtx
}

func 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 cmd {
    case "-i":
        fdix = 0
        mode = os.O_RDONLY
    case "-o":
        fdix = 1
        mode = os.O_RDWR | os.O_CREATE
    case "-a":
        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, _ = tgshell1(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
}

func gshellv(gshCtx GshContext, argv []string) (_ GshContext, fin bool) {
    fin = false

    if gshCtx.CmdTrace { fmt.Printf("--I-- gshellv((%d))\n",len(argv)) }
    if len(argv) <= 0 {
        return gshCtx, false
    }
    for ai := 0; ai < len(argv); ai++ {
        argv[ai] = strsubst(argv[ai])
    }
    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.Printf("--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 == "-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 == "-o" || cmd == "-a" || cmd == "-s":
    redirect(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 == "#define":
case cmd == "echo":
    echo(argv,true)
case cmd == "env":
    env(argv)
case cmd == "eval":
    eval(argv,true)
case cmd == "exec":
    gshCtx, _ = excommand(gshCtx, true,argv[1:])
    // should not return here
case cmd == "exit" || cmd == "quit":
    // write Result code EXIT to 3>
    return gshCtx, true
case cmd == "-find":
    find(argv)
case cmd == "fork":
    // mainly for a server
case cmd == "-gen":
    gen(gshCtx, argv)
case cmd == "go":
    xGo(gshCtx, argv)
case cmd == "history" || cmd == "hi": // hi should be alias
    gshCtx = xHistory(gshCtx, argv)
case cmd == "jobs":
    xJobs(gshCtx,argv)
case cmd == "nop":
case cmd == "open":
    gshCtx = xOpen(gshCtx,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 == "set":
    // set name ...
case cmd == "serv":
    httpServer(gshCtx,argv)
case cmd == "sleep":
    sleep(gshCtx,argv)
case cmd == "time":
    gshCtx, fin = xTime(gshCtx,argv)
case cmd == "pwd":
    xPwd(gshCtx,argv);
case cmd == "ver" || cmd == "-ver":
    fmt.Printf("%s\n",VERSION);
case cmd == "where":
    // data file or so?
case cmd == "which":
    which("PATH",argv);
default:
    gshCtx, _ = excommand(gshCtx, false,argv)
}
return gshCtx, fin
}

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

func tgshelll(gshCtx GshContext, gline string) (gx GshContext, xfin bool) {
    start := time.Now()
    gshCtx, fin := gshelll(gshCtx,gline)
    end := time.Now()
    elps := end.Sub(start);
    fmt.Printf("--I-- " + time.Now().Format(time.Stamp) + " (%d.%09ds)\n",
        elps/1000000000,elps%1000000000)
    return gshCtx, fin
}

```

```

}
func Ttyid() (int) {
    fi, err := os.Stdin.Stat()
    if err != nil {
        return 0;
    }
    //fmt.Printf("Stdin: %v Dev=%d\n",
    //    fi.Mode(), fi.Mode() & os.ModeDevice)
    if (fi.Mode() & os.ModeDevice) != 0 {
        stat := syscall.Stat_t{};
        err := syscall.Fstat(0, &stat)
        if err != nil {
            //fmt.Printf("--I-- Stdin: (%v)\n", err)
        } else {
            //fmt.Printf("--I-- Stdin: rdev=%d %d\n",
            //    stat.Rdev & 0xFF, stat.Rdev);
            //fmt.Printf("--I-- Stdin: tty=%d\n", stat.Rdev & 0xFF);
            return int(stat.Rdev & 0xFF)
        }
    }
    return 0
}
func ttyfile(gshCtx GshContext) string {
    //fmt.Printf("--I-- GSH_HOME=%s\n", gshCtx.GshHomeDir)
    ttyfile := gshCtx.GshHomeDir + "/" + "gsh-tty" +
        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
}
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("Proc ERROR (%s)\n", nil)
            for ; ; {
            }
        }
        //stat, err := proc.Wait()
        proc.Wait()
        buff := make([]byte, LINESIZE)
        count, err := tty.Read(buff)
        //_, err = tty.Read(buff)
        //fmt.Printf("--D-- getline (%d)\n", count)
        if err != nil {
            if ! (count == 0) { // && err.String() == "EOF" } {
                fmt.Printf("--E-- getline error (%s)\n", err)
            }
        } else {
            //fmt.Printf("--I-- getline OK \"%s\"\n", buff)
        }
        tty.Close()
        return string(buff[0:count])
    } else {
        // if isatty {
        //    fmt.Printf("!!%d", hix)
        //    fmt.Print(PROMPT)
        // }
        reader := bufio.NewReaderSize(os.Stdin, LINESIZE)
        line, _, _ := reader.ReadLine()
        return string(line)
    }
}
//
// $USERHOME/.gsh/
// gsh-rc.txt, or gsh-configure.txt
// gsh-history.txt
// gsh-aliases.txt // should be conditional?

```

```

//
func gshSetupHomedir(gshCtx GshContext) (GshContext, bool) {
    homedir, err := os.UserHomeDir()
    if err != nil {
        fmt.Printf("--E-- You have no UserHomeDir (%v)\n",err)
        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,err)
            return gshCtx, true
        }
        fmt.Printf("--I-- Created %s\n",gshhome)
    }
    gshCtx.GshHomeDir = gshhome
    return gshCtx, false
}

func setupGshContext() (GshContext,bool){
    gshPA := syscall.ProcAttr {
        "", // the staring directory
        os.Environ(), // environ[]
        []uintptr{os.Stdin.Fd(),os.Stdout.Fd(),os.Stderr.Fd()},
        nil, // OS specific
    }
    cwd, _ := os.Getwd()
    gshCtx := GshContext {
        cwd, // StartDir
        "", // GetLine
        []GchdirHistory { {cwd,time.Now()} }, // ChdirHistory
        gshPA,
        []GCommandHistory{}, //something for invokation?
        GCommandHistory{}, // CmdCurrent
        false,
        []int{},
        syscall.Rusage{},
        "", // GshHomeDir
        Ttyid(),
        false,
    }
    err := false
    gshCtx, err = gshSetupHomedir(gshCtx)
    return gshCtx, err
}

func script(gshCtxGiven *GshContext) (_ GshContext) {
    gshCtx,err0 := setupGshContext()
    if err0 {
        return gshCtx;
    }
    //fmt.Printf("--I-- GSH_HOME=%s\n",gshCtx.GshHomeDir)
    //resmap()
    gsh_getlinev, with_exgetline :=
        which("PATH",[]string{"which","gsh-getline","-s"})
    if with_exgetline {
        gsh_getlinev[0] = toFullpath(gsh_getlinev[0])
        gshCtx.GetLine = toFullpath(gsh_getlinev[0])
    }else{
        fmt.Printf("--W-- No gsh-getline found. Using internal getline.\n");
    }

    prevline := ""
    skipping := false
    for hix := 1; ; {
        gline := getline(gshCtx,hix,skipping,with_exgetline,gsh_getlinev,prevline)
        if skipping {
            if strings.Index(gline,"fi") == 0 {
                fmt.Printf("fi\n");
                skipping = false;
            }else{
                //fmt.Printf("%s\n",gline);
            }
            continue
        }
        if strings.Index(gline,"if") == 0 {
            //fmt.Printf("--D-- if start: %s\n",gline);
            skipping = true;
            continue
        }
        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()
    }
}

```

```

var rusage1 = syscall.Rusage {}
var rusage2 = syscall.Rusage {}
syscall.Getrusage(syscall.RUSAGE_SELF, &rusage1)
xgshCtx, fin := tgshell1(gshCtx, gline)
syscall.Getrusage(syscall.RUSAGE_SELF, &rusage2)
RusageSub(&rusage2, &rusage1)
ghist.Rusage = rusage2
gshCtx = xgshCtx
ghist.EndAt = time.Now()
ghist.CmdLine = gline

/* 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() {
    gshCtx := script(nil)
    gshell1(gshCtx, "time")
}
// TODO:
// - 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
// - all commands have its subucomand after "---" symbol
// - filename expansion by "-find" command
// - history of ext code and output of each commoand
// - "script" output for each command by pty-tee or telnet-tee
// - $BUILTIN command in PATH to show the priority
// - "?" symbol in the command (not as in arguments) shows help request
// - searching command with wild card like: which ssh-*
// - longformat prompt after long idle time (should dismiss by BS)
//---END--- (^-^)/

```