```
//

// 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"
	"fmt"
	"os"
	"time"
	"syscall"
	"go/types"
	"go/token"
	"net"
)

var VERSION = "gsh/0.0.4 (2020-0808a)"
var LINESIZE = (8*1024)
var PATHSEP = ":" // should be ";" in Windows
var PROMPT = "> "

func env(argv []string) {
	env := os.Environ()
	for _, v := range env {
		fmt.Printf("%v\n",v)
	}
}
func which(path string, show bool) (xfullpath string, itis bool){
	pathenv, found := os.LookupEnv("PATH")
	if found {
		dirv := strings.Split(pathenv,PATHSEP)
		for _, dir := range dirv {
			fullpath := dir + "/" + path
			fi, err := os.Stat(fullpath)
			if err != nil {
				fullpath = dir + "/" + path + ".go"
				fi, err = os.Stat(fullpath)
			}
			if err == nil {
				fm := fi.Mode()
				if fm.IsRegular() {
					if show {
						fmt.Printf("%s\n",fullpath)
					}
					return fullpath, true
				}
			}
		}
	}
	return "", false
}
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(gshPA syscall.ProcAttr, exec bool, argv []string) (ret int) {
	fullpath, itis := which(argv[0],false)
	if itis == false {
		return -1
	}
	if 0 < strings.Index(fullpath,".go") {
		nargv := argv // []string{}
		gofullpath, itis := which("go",false)
		if itis == false {
			fmt.Printf("-- Go not found\n")
			return -1
		}
		nargv = []string{ gofullpath, "run", fullpath }
		fmt.Printf("-- %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)
			syscall.Wait4(pid,nil,0,nil);
		}
	}else{
		if exec {
			syscall.Exec(fullpath,argv,os.Environ())
		}else{
			pid, _ := syscall.ForkExec(fullpath,argv,&gshPA)
			//fmt.Printf("[%d]\n",pid); // '&' to be background
			syscall.Wait4(pid,nil,0,nil);
		}
	}
	return 0
}
```

```
func sleep(gshPA syscall.ProcAttr, 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 ns\n",duration)
	time.Sleep(d)
	if 0 < len(argv[2:]) {
		gshellv(gshPA, argv[2:])
	}
}
func repeat(gshPA syscall.ProcAttr, 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(gshPA, argv[2:])
			end := time.Now()
			elps := end.Sub(start0);
			if( 1000000000 < elps ){
				fmt.Printf("(repeat#%d %v)\n",ri,elps);
			}
		}
	}
}

func gen(gshPA syscall.ProcAttr, argv []string) {
	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("-- 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(gshPA syscall.ProcAttr, inTCP bool, argv []string) {
	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(gshPA, 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(gshPA, argv[2:])
		gshPA.Files[1] = savfd
		file.Close()
		conn.Close()
	}
}
func saccept(gshPA syscall.ProcAttr, inTCP bool, argv []string) {
	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(gshPA, argv[2:])
                gshPA.Files[0] = savfd

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

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

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

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

// empty line command
func pwd(gshPA syscall.ProcAttr){
        // execute context command, pwd + date
        // context notation, representation scheme, to be resumed at re-login
        cwd, _ := os.Getwd()
        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")
}
func gshellv(gshPA syscall.ProcAttr, argv []string) (fin bool) {
        //fmt.Printf("-- gshellv((%d))\n",len(argv))
        if len(argv) <= 0 {
                return false
        }
        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 cmd == "-ot" {
		sconnect(gshPA, true, argv)
		return false;
	}
	if cmd == "-ou" {
		sconnect(gshPA, false, argv)
		return false;
	}
	if cmd == "-it" {
		saccept(gshPA, true , argv)
		return false;
	}
	if cmd == "-iu" {
		saccept(gshPA, false, argv)
		return false;
	}
	if cmd == "-i" || cmd == "-o" || cmd == "-a" || cmd == "-s" {
		if len(argv) < 2 {
			return false
		}
		fdix := 0;
		mode := os.O_RDONLY;
		if cmd == "-i" {
		}
		if cmd == "-o" {
			fdix = 1;
			mode = os.O_RDWR | os.O_CREATE;
		}
		if cmd == "-a" {
			fdix = 1;
			mode = os.O_RDWR | os.O_CREATE | os.O_APPEND;
		}
		f, err := os.OpenFile(argv[1], mode, 0600)
		if err != nil {
			fmt.Printf("%s\n",err)
			return false
		}
		savfd := gshPA.Files[fdix]
		gshPA.Files[fdix] = f.Fd()
		fmt.Printf("-- Opened [%d] %s\n",f.Fd(),argv[1])
		gshellv(gshPA, argv[2:])
		gshPA.Files[fdix] = savfd
		return false
	}
	if cmd == "call" {
		excommand(gshPA, false,argv[1:])
		return false
	}
	if cmd == "echo" {
		echo(argv,true)
		return false
	}
	if cmd == "env" {
		env(argv)
		return false
	}
	if cmd == "eval" {
		eval(argv,true)
		return false
	}
	if cmd == "exec" {
		excommand(gshPA, true,argv[1:])
		return false // should exit
	}
	if cmd == "exit" || cmd == "quit" {
		// write Result code EXIT to 3>
		return true
	}
	if cmd == "fork" {
```

```
                // mainly for a server
                return false
        }
        if cmd == "-gen" {
                gen(gshPA, argv)
                return false;
        }
        if cmd == "nop" {
                return false
        }
        if cmd == "pstitle" {
                // to be gsh.title
        }
        if cmd == "repeat" { // repeat cond command
                repeat(gshPA,argv)
                return false
        }
        if cmd == "set" { // set name ...
                return false;
        }
        if cmd == "sleep" {
                sleep(gshPA,argv)
                return false;
        }
        if cmd == "-ver" {
                fmt.Printf("%s\n",VERSION);
                return false
        }
        if cmd == "pwh" {
                pwd(gshPA);
                return false
        }
        if cmd == "which" {
                which(argv[1],true);
                return false
        }
        excommand(gshPA, false,argv)
        return false
}
func gshelll(gshPA syscall.ProcAttr, gline string) (rfin bool) {
        argv := strings.Split(string(gline)," ")
        fin := gshellv(gshPA,argv)
        return fin
}
func tgshelll(gshPA syscall.ProcAttr, gline string) (xfin bool) {
        start := time.Now()
        fin := gshelll(gshPA,gline)
        end := time.Now()
        elps := end.Sub(start);
        fmt.Printf("--(%d.%09ds)\n",elps/1000000000,elps%1000000000)
        return fin
}
func ttyfile() string {
        return "gsh.ttyline"
}
func ttyline() (*os.File){
        file, err := os.OpenFile(ttyfile(), os.O_RDWR|os.O_CREATE|os.O_TRUNC,0600)
        if err != nil {
                fmt.Printf("cannot open %s (%s)\n",ttyfile(),err)
                return file;
        }
        return file
}
func getline(hix int, with_exgetline bool, prevline string) (string) {
        if( with_exgetline ){
                //var xhix int64 = int64(hix); // cast
                newenv := os.Environ()
                newenv = append(newenv, "GSH_LINENO="+strconv.FormatInt(int64(hix),10) )

                tty := ttyline()
                tty.WriteString(prevline)
                Pa := os.ProcAttr {
                        "", // start dir
                        newenv, //os.Environ(),
                        []*os.File{os.Stdin,os.Stdout,os.Stderr,tty},
                        nil,
                }
                proc, err := os.StartProcess("gsh-getline",[]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("-- getline (%d)\n",count)
                if err != nil {
                        if ! (count == 0) { // && err.String() == "EOF" ) {
                                fmt.Printf("-- getline error (%s)\n",err)
                        }
                }else{
                        //fmt.Printf("-- 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)
        }
}
func main() {
        gshPA := syscall.ProcAttr {
                "", // the staring directory
                os.Environ(), // environ[]
                []uintptr{os.Stdin.Fd(),os.Stdout.Fd(),os.Stderr.Fd()},
                nil, // OS specific
        }
        //resmap()
        _, with_exgetline := which("gsh-getline",false)
        if with_exgetline == false {
                fmt.Printf("Note: No gsh-getline found. Using internal getline.\n");
        }
        prevline := ""
        for hix := 1; ; {
                gline := getline(hix,with_exgetline,prevline)
                fin := tgshelll(gshPA,gline)
                if fin {
                        break;
                }
                if len(gline) == 0 {
                        pwd(gshPA);
                        continue;
                }
                prevline = gline;
                hix++;
        }
}
// 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
//---END--- (^-^)/
```