```
//

// gsh - Go lang based Shell
// (c) 2020 ITS more Co., Ltd.
// 2020-0807 created by SatoxITS (sato@its-more.jp)
//
// Reference: https://golang.org/pkg/
//
package main // gsh main

import (
        "bufio"
        "strings"
        "strconv"
        "fmt"
        "os"
        "time"
        "syscall"
        "go/types"
        "go/token"
)

var VERSION = "gsh/0.0.2 (2020-0807b)"
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, 0)
        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
        }
        for ri,_ := strconv.Atoi(argv[1]); 0 < ri; ri-- {
                if 0 < len(argv[2:]) {
                        gshellv(gshPA, argv[2:])
                }
        }
}
func gshellv(gshPA syscall.ProcAttr, argv []string) (fin bool) {
        if len(argv) <= 0 {
                return false
        }
        cmd := argv[0]
        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;
		}
		if cmd == "-s" {
			// bi-directional, source/sync, maybe socket
		}
		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 == "nop" {
		return false
	}
	if cmd == "-ver" {
		fmt.Printf("%s\n",VERSION);
		return false
	}
	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 == "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 main() {
	gshPA := syscall.ProcAttr {
		"",
		os.Environ(),
		[]uintptr{os.Stdin.Fd(),os.Stdout.Fd(),os.Stderr.Fd()},
		nil,
	}
	resmap()
	for hix := 0; ; hix++ {
		fmt.Printf("%d",hix)
		fmt.Print(PROMPT)
		reader := bufio.NewReaderSize(os.Stdin,LINESIZE);
		gline, _, _ := reader.ReadLine()

		fin := tgshelll(gshPA,string(gline))
		if fin {
			break;
		}
	}
}
//---END--- (^-^)/
```