Massey University

159.251 Software Design and Construction

Command Line Interfaces (CLIs) and Scripting

Recap - Operating Systems

OS Functions

- An OS manages resources, it’s a bridge between hardware and software and it provides file system services
- Provides a User interface (Graphical User Interface (GUI), Command line (CLI) for launching programs and managing files
- Provides system services to application software

Components

- PC Boot sequence (from power up to logon screen)
- Core OS (memory, process and device management), device drivers, user interface (GUI, CLI), file managers, utilities and application software.
- Configuring settings (network, user settings (names/password/group), file system security)

File systems

- File system layouts (system, applications, configuration and user directories)
- Files (named block of data)
- Directories (hierarchical structure for organizing files)

File systems

Formats: ext4 (Linux), hfs (MacOS), ntfs/fat32 (Windows)

A file’s full path identifies its exact location

In Windows: c:\Users\joe\file.txt

In Linux /home/joe/file.txt

Absolute paths and relative paths (relative paths from a working directory, . denotes the current directory and .. the parent directory of the current directory)

If /home is the working directory, then ./joe/file.txt is the relative path, .. is the parent of the working directory, i.e. .. is /home

Security

Needed for multiuser/network OSes

Users assigned accounts/passwords

Two classes of users (normal and admin)

Permissions to control user access to files (Windows, Linux)

Environment

Used for communicating parameters or settings for applications

E.g. PATH environment variable

HOME variable in Linux and USERPROFILE in Windows

Query environment using env, on Linux and set in Windows

Expanding environment variable: $PATH (Linux) %PATH% (Windows)

Accessing Environment variables in Java

```
Map<String, String> env = System.getenv();

String path = env.get(“PATH”);
```

Command line

Console, physical terminal

Terminal, a program with a text interface

Shell or Command Prompt, a program that runs in a terminal that interprets commands (OS-specific)

Linux and OS X: bash (bourne again shell)

Windows: bash (Windows Subsystem for Linux), PowerShell, Command Prompt or cmd

Unlike Graphical User Interfaces that share concepts such as menus/context sensitive menus, command line interfaces require knowledge of command names

Online help

- Windows

`<command> /?` (Usage: `dir /?`)

`help`

- Linux / OS X

Manual pages: `man <command>` (Usage: `man ls`)

cmd --help (Usage: ls --help)

tldr (https://tldr.sh/) (Usage: tldr ls)

- `Auto completion and command history facility`

How to start a command line session

Launch or run cmd in Windows

Launch Terminal in OS X and Linux (bash)

Start remote sessions on Linux servers (ssh or Putty client programs)

.profile and .bashrc executed on login

List running processes

Windows

```
tasklist
```

Linux/OS X

```
ps
```

Output

```
 PID (Process Identifier), name, user
```

Working with processes

Ctrl+C interrupts a running program

On Linux/ OS X

To terminate a process:

kill <PID>

To run a program in the background, append the command with &

jobs # to view processes started in the session

fg   # brings job to the foreground

Ctrl+Z stops running job

Display current working directory

Windows

```
cd
```

Linux / OS X

```
pwd
```

Output

```
 Current directory
```

List files and directories

Windows

dir

Linux / OS X

ls

Output

```
 List
```

```
Shells expand wildcards in commands. E.g. ls *.pdf or dir
*.pdf
```

Find files

Windows

dir /s

Linux / OS X

find .

Copy files

Windows

Copy <source file> destination_file

Linux / OS X

cp source target

Rename files

Windows

rename file newfile
Rename dir newdir

Linux/ OS X

mv file newfile
mv dir newdir

Change current directory

Windows

`cd \Users` and change drive with the command C: for drive C

Linux / OS X

cd /home or cd /Users

Output: Changes to the specified directory

- Shortcut for home: ~ Current directory: . and for parent directory: ..

Create a directory

Windows

mkdir tmpdir

Linux / OS X

mkdir tmpdir

Recursively create nested directories: mkdir -p tmpdir/childdir

Output

```
 Creates the specified directory
```

Delete files and directories

Windows

del tmp.txt

rmdir /s tmpdir

Linux / OS X

rmdir tmpdir

Recursively created nested directories: mkdir -p tmpdir/childdir

Output

```
Deletes specified directories
```

Security Permissions

Windows

Use GUI, Select file/folder properties context-sensitive menu, navigate to the Security tab and set the Access Control Entries

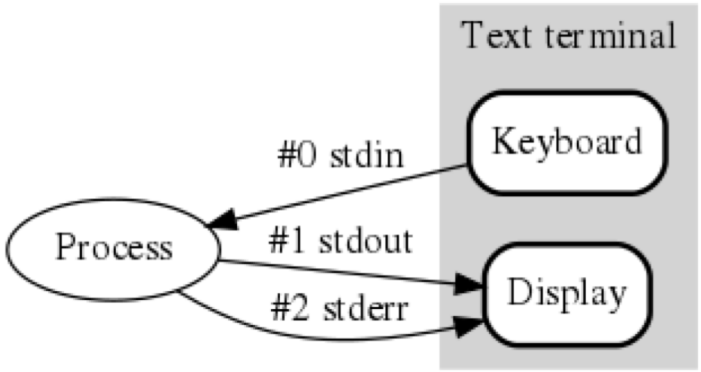
Linux

Permissions (**r**ead, **w**rite, e**x**ecute) for a file/directory can be set for a **u**ser, the **g**roup that owns the file and for **o**thers

chmod u=rwx,g=rx,o=r myfile

Standard Input/Output Streams

Communication channels for a program and its environment



Standard output, Standard error and Standard Input

In Java: use System.out.println System.err.println and System.in.read to communicate via these channels

Redirection

Reroute input or output channels to a file. To redirect standard output to a file:

Use > to replace the file with the output

Use >> to append the output to the file

dir > output.txt

Redirect standard error using 2>&1

This takes file descriptor #2 (stderr) and redirects it to file descriptor #1 (stdout)
e.g. someCmd >> /tmp/someCmd.log 2>&1

Redirection

Programs can be fed input by redirecting standard input to a file

e.g.

cat < test.txt

sort < test.txt

Composing programs

Unix Philosophy

“Write programs that do one thing and do it well. Write programs to work together. Write programs to handle text streams, because that is a universal interface.”

- A Quarter Century of Unix

Multitude of small programs that do one thing. e.g.

sort, cut, head, tail, rev, tr, sed, wc

How to compose them together?

Pipes

The output of one program can be "piped" into the input of the next using | e.g.

cat takes an input file and output it (to stdout)
sort takes a file (or stdin), sorts, output lines (to stdout)
Pipe testfile.txt into sort and output the lines

cat testfile.txt | sort

Searching text

Command line tasks involve processing text

Need to find or filter out specific information.

An example is you might want to extract all lines in a file with the string “the”

You can use the grep utility for this task

On Linux/OS X the **grep command** takes a search pattern and a file as arguments.

On Windows **findstr** is an option to search for text

It can also process standard input if a file is not given as an argument

Grep

File.txt

```
This is the first line
The second line
The end of the file
This file is complete
```

grep “the” file.txt prints the first and third lines

grep -r “the” * recursively searches all finds under current directory

Can extend basic search patterns with **regular expressions**

Regular expressions

A sequence of characters that defines a search pattern

Metacharacter	**Description**
.	The dot metacharacter matches any character.
[ ]	Matches a single character in the bracket. E.g. [abc] matches either a, b or c. [a-z] matches any character from a to z. [0-9] any digit
[^]	Matches any character not in bracket. [^abc] matches any character but a, b or c
^ and $	^ matches beginning of line. $ matches the end
*	Matches the preceding character in the pattern zero or more times. E.g. ab*c matches ac, abc, abbc
\{m\}	Matches element m times. b\{2\} matches bb

Regular expressions

Metacharacter	Description
\{m,\}	Matches element at least m times
\{m,n\}	Matches element at least m and at most n times
\(\)	Defines a subexpression. While ab* matches a, ab, abb ... \(ab\)* matches ab, abab, abababab ...

To match special characters you must escape them. E.g. grep “a*” will search for “a*” and will not process the escaped * as a metacharacter.

With extended regular expressions (enabled using grep -E) escaped metacharacters can be specified without the backslashes
e.g. grep -E “(ab)*” instead of grep “\(ab\)*”

Scripting to automate command line tasks

Windows

- Batch file scripts
- Bash scripts (Made possible through WLS)
- PowerShell Scripts (powerful, well integrated with the Windows ecosystem but more suitable for configuration management of servers)

Linux / OS X

- Bash scripts

Other alternatives: Use a scripting language such as Python

Exit codes

Programs can communicate a value known as the exit code to the parent process

In Windows this can be accessed via the errorlevel variable

In Linux/OS X, the variable that stores the exit code is $?

Java programs return the exit code through System.exit

E.g. System.exit(0)

Normally, 0 signals success and 1 or higher for failed execution

Windows Batch files

Commands issued at the prompt can be compiled and saved in a batch file.

E.g. the following lines can be saved in script.cmd and executed at the prompt as cmd files are executable

```
REM comment
ECHO Deleting file
CD %USERPROFILE%
DEL tmp.txt
```

More Windows batch scripting features

- Variables
- Command line parameters
- If/then statements
- Loops

Variables

SET a=Test

Convention to use lowercase variable names and leave uppercase names for environment variables

SETLOCAL command at the beginning of a batch script will inherit all current variables from the master environment/session.

ENDLOCAL command will restore any environment variables present before the SETLOCAL was issued.

Parameter variables

Command line parameters to the batch script can be referred to using %0, %1,, %9 within the batch script

This syntax can be extended to obtain more information from a parameter

If the variable %1 ‘s value is a filename, the batch script can obtain the fully qualified path name using %~f1 - C:\Users\joe\scripts\MyFile.txt

More on these extensions at https://ss64.com/nt/syntax-args.html

If/then statement

```
IF EXIST filename.txt (
    Echo deleting filename.txt
    Del filename.txt
 ) ELSE (
    Echo The file was not found.
 )
```

In addition to conditions over files, string equality comparisons are also supported

https://ss64.com/nt/if.html

Batch script

```
@echo off

findstr %1 test.txt

if %errorlevel% leq 0 (

echo "Successfully found matching text"

) else (

echo "Failed to find matching text"

)
```

Loops

```
@ECHO OFF

REM Prints names of all files in working directory


FOR %%A IN (*) DO (
	echo %%A
)
```

Bash shell scripts

Shell commands that can be issued at the prompt can be saved in a file for execution when required

The first line of the script should be

#!/bin/bash

The script file should be set as an executable file by changing its permissions :

chmod u+x script.sh

Bash shell scripts

```
#!/bin/bash
# comment
echo Deleting file
cd $HOME
rm tmp.txt
```

More Bash scripting features

- Variables
- If/then statements
- Loops

Variables

var=Hello
*export var # **shell variables** must be exported for them to be visible to subsequent processes as **environment variables***
Parameter variables

A script can refer to $1, $2.. as parameter variables

Setting variables and using them

variable=”/home/joe”
ls $variable

Command substitution
variable=$(ls) # variable is populated with results of the ls command

If/then statements

```
if [ <test> ]
then
   <commands>
fi
```

Tests can check for things like the existence of a file or string equality

```
If [ -e foo.txt ]
then
  echo “file exists”
fi
```

Loops

```
for dir in *
do

 echo $dir

end

for d in *; do echo $d; done     # single line
```

Resources

Setting up the Windows Linux Subsystem

https://docs.microsoft.com/en-us/windows/wsl/install-win10

Experiment with regular expressions

https://regexr.com/