The shell

COMS10012 / COMSM0085

Software Tools

The shell

seis-f5-d0.services.bris.ac.uk - PuTTY

```
csxdb@it075734:~/vagrant$ls
micro        sample-data.sql   user          Vagrantfile.mariadb
sampledata   secure-setup.sql  Vagrantfile   Vagrantfile.original
csxdb@it075734:~/vagrant$ls -l sampledata/
total 8
drwxr-xr-x. 2 csxdb cosc 4096 Jul 22 11:47 census
drwxr-xr-x. 2 csxdb cosc 4096 Jul 22 11:47 elections
csxdb@it075734:~/vagrant$cat sampledata/elections/elections-2014.csv | grep Bris
lington | wc -l
12
csxdb@it075734:~/vagrant$
```

Terms

shell

terminal

console

command line

(command) prompt

xterm

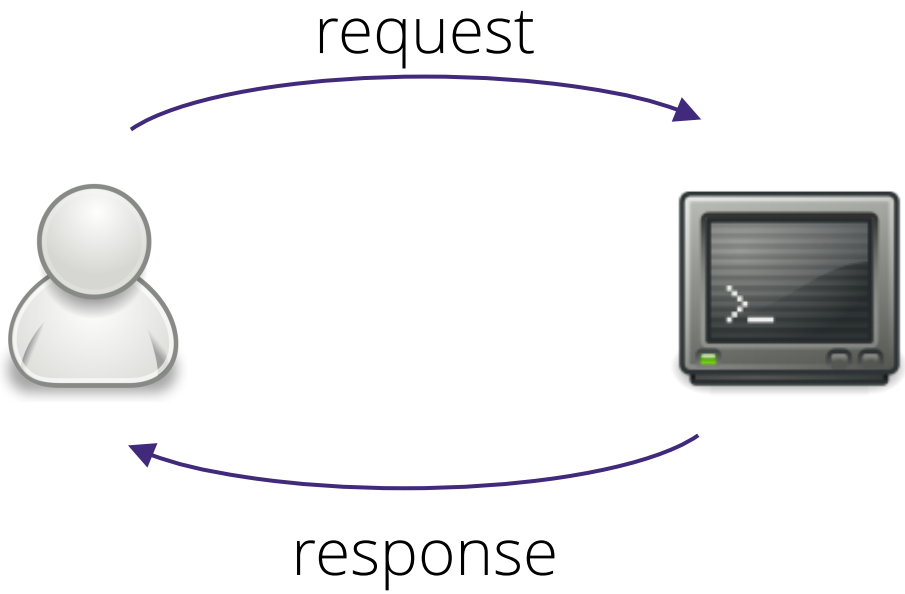
rxvt

konsole

(gnome)-terminal

putty (Windows)

shell workflow



prompt

$	You are in a shell, most likely POSIX (sh) compatible.
#	You are in a root shell. With great power comes great responsibility.
%	You are probably in the C shell.
>	You are on a continuation line e.g. inside a string.

shell tricks

TAB: complete command or filename

DOUBLE TAB: show list of possible completions

UP/DOWN: scroll through history

^R text: search history for command

builtins

```
$ which ls
/bin/ls
$ which cd
$
```

options and conventions

```
$ ls
file1          file2
$ ls -l
-rwx------     1 vagrant ... 40 ... file1
-rwxr-----     1 vagrant ... 80 ... file2
$ ls -a
.          ..          file       file2
```

help

```
$ ls --help
BusyBox v1.30.1 multi-call binary.

Usage: ls [-1AaCxdLHRFplinshrSXvctu] [-w
WIDTH] [FILE]...

List directory contents

  -1      One column output
  -a      Include entries which start with
.
  ...     ...
```

manuals

```
$ man [SECTION] COMMAND
```

- On lab machines: fairly user-friendly manual.
- On alpine: programmer's manual.

Section 1 is shell commands, section 2 system calls, section 3 the C library etc.

e.g. `man 1 printf` and `man 3 printf` are different.

shell expansion

shell expansion

Separation of responsibility:

- shell deals with expanding pattern
- program deals with its arguments

shell expansion

`*` all filenames in current scope
e.g `a*` is filenames starting with a etc.

`?` single character in filename
e.g. `image???.jpg` matches
`image001.jpg`

`[ab]` single character in list
e.g. `image[0-9].jpg`

`$` variable name expansion

shell quoting

`"double quotes"` turn off pattern matching
keeps variable interpolation and backslashes on

`'single quotes'` turn off everything

`\*, \?, \[, \$` do not treat as pattern

example

`cp [-rfi] SRC... DEST` copy files

- **`-r`** recursive
- **`-f`** overwrite readonly
- **`-i`** ask before overwriting (interactive)

`mv [-nf] SRC... DEST` move files

- **`-n`** no overwrite
- **`-f`** force overwrite

examples

```
$ cp index.html style.css web
$ cp * web
```

in empty folder:

```
$ cp * web
cp: can't stat '*': No such file or
directory
```

In the broadest terms, a shell is a command-line interface (CLI) program that provides users a way to interact with the operating system. It acts as an intermediary between the user & the kernel (the core of the OS), allowing users to execute commands, run programs, manipulate files & perform various tasks through textual commands & scripts.

This lecture is an overview on Shell Expansion.

- Whitespace

  Whitespace is ignored unless its part of a string literal, then it considered - this also splits different words on command line.

  e.g

  ./arguments one two → [./arguments] [one] [two]

  ./arguments "one two" → [./arguments] [one two]

- touch [FILENAME]

  creates a file.

This means if you forget quotations when specifying file names & the file name contains spaces, only the first word is considered.

Shell Variables

In the shell:

VARIABLE = VALUE sets a variable to a value & $VARIABLE retrieves that value

e.g in a shell script:

```
p = arguments
gcc -Wall $p.c -o $p
```

⟹ gcc -Wall arguments.c -o arguments

This can cause issues if you use bad values.

For example

```
prog="spaced name"
gcc -Wall $prog.c -o $prog
```

⇓

gcc -Wall spaced name.c -o spaced name

error

Important note is you need to set variables on seperate lines to the commands that call them

e.g

file=arguments gcc -Wall "$file.c" -o "$file"

This fails because the shell first replaces variables ($file) before commands are executed ∴ $file is undefined at runtime

So it is good practice to double quote expressions containing variables to make scripts robust against this.

```
gcc -Wall "$prog.c" -o "$prog"
```

⇓

gcc -Wall "spaced name.c" -o "spaced name" (no error)

note, using spaces in names is dumb don't do it.

find files

```
$ find DIR [EXPRESSION]
```

find all files in directory (recursively)
that match an expression

e.g. `find . -name "a*"`