2. Shell Tools and Scripting

Arguments

Arguments are separated by whitespace
So when a argument contains whitespace

```
echo Hello\ World
echo "Hello World"
```

Variable `$`

```
foo=bar
echo $foo       # bar
```

String in bash

Double quotes v.s. Single quote

1. Define a *string* (no difference)

```
echo "Hello"      # Hello
echo 'World'      # World
```

2. include a *variable*
   in single quote, won't be replacing

```
echo "Value is $foo"     # Value is bar
echo 'Value is $foo'     # Value is $foo
```

Control flow

```
~/m/tools ❯❯❯ vim mcd.sh
~/m/tools ❯❯❯ source mcd.sh
~/m/tools ❯❯❯ mcd test
```

for loop, while loop, define functions

1. Access a function
   `vim mcd.sh`

```
mcd (){
    mkdir -p "$1"            # sequential execution
    cd "$1"                  # "$1" means the first argument, as argv in C
}
```

2. Execute this script and load it
   `source mcd.sh`
   `mcd test`

Reserved commands

`$0` --> the name of the script
`$2` - `$9` --> the second through the ninth arguments
`$?` --> the **error code** from the previous command
`$_` --> the **last argument** of the previous command
`$#` --> the number of arguments that are giving to the command
`$$` --> the process ID of this command that is running

Error message

exit code

`0` --> (everything goes fine, there are no errors)
`1` --> error code

```
~/m/tools ❯❯❯ echo "Hello"
Hello
~/m/tools ❯❯❯ echo $?
0
~/m/tools ❯❯❯ grep foobar mcd.sh
~/m/tools ❯❯❯ echo $?
1
```

Conditions

1. two commands connected by the `OR` operator

```
false || echo "Oops fail"              # Oops fail
true || echoe "Will be no printed"     #
```

(bash will execute the first one, **if the first one didn't work, then** it gonna execute the second one)

2. `AND` operator

```
false && echo "This will not print"     #
true && echo "Things went well"         # Things went well
```

(**only execute the second one when the first one ran without errors**)

3. concatenate commands using `;` in the same line

```
false ; echo "This will always print"
```

Command Substitution

Sudo Permission

`sudo !!` -- will replace the previous command in

```
~/m/tools ❯❯❯ mkdir /mnt/new
mkdir: /mnt/new: Permission denied
~/m/tools ❯❯❯ sudo !!
```

--> ask for password
--> expand `sudo !!` to `sudo mkdir /mnt/new`

Expanding to a string

```
foo=$(pwd)              # Getting the output of a command into a variable
echo $foo               # /Users/ishtar
```

1. getting the output of the `pwd` command
   (`pwd` --> printing the Present Working Directory)
2. storing that into the `foo` variable

```
echo "We are in $(pwd)"      # We are in /Users/ishtar"
```

Process Substitution

(input from `file`, instead of `stdin`)

1. `ls` the directory, put the content into a temporary file, doing the same thing for the parent folder, and **concatenate** both files
   `cat <(ls) <(ls ..)`

Globbing(通配符) --> Arguments file name expansion

1. `*`

```
~/m/tools ❯❯❯ ls
example.sh  image.png  mcd.sh  project1  project2  script.py  test
~/m/tools ❯❯❯ ls *.sh
```

2. `?` : expand to only a **single one**

```
~/m/tools ❯❯❯ mkdir project42
~/m/tools ❯❯❯ ls project?
project1:
src

project2:
src
```

3. `{}` : combine
   e.g. want to convert one format to another, it will **automatically expanded** into the above line

```
~/m/tools ❯❯❯ convert image.png image.jpg
~/m/tools ❯❯❯ convert image.{png,jpg}
```

```
touch foo{,1,2,10}
# expanded into touch foo foo1 foo2 foo10


touch project{1,2}/src/test/test{1,2,3}.py


mkdir foo bar
touch {foo,bar}/{a..j}

touch foo/x bar/y
diff <(ls foo) <(ls bar)
```

- `diff`
  compare files line by line
  In this case, it is being used to compare the output of two subcommands.
- `<(...)`
  process substitution
  It takes the *output* of the command inside the parentheses and makes it available to `diff` as if it were a file.
- `ls foo` : Lists the contents of the directory `foo`.
- `ls bar` : Lists the contents of the directory `bar`.

Shell Scripts

Example

`vim example.sh`

this script takes one or more file names as arguments. For each file, it checks if the file contains the string "foobar". If not, it prints a message to the terminal and adds a line (`# foobar`) to the end of the file.

```
#!/bin/bash
echo "Starting program at $(date)"   # date will be substituted

echo "Running program $0 with $# arguments with $$"
```

```
for file in "$@"; do
    grep foobar "$file" > /dev/null 2> /dev/null
    # when pattern is not found, grep has exit status
    # redirect the STDOUT and STDERR to a null register
    if [[ "$?" -ne 0]]; then
        echo "File $file does not have foobar"
        echo "# foobar" >> "$file"
    fi
done
```

Notes

1. **shebang**
   is the way that the shell will know how to execute this script, located at `/bin/bash`.

```
#!/bin/bash           # Give the path to shell where that thing lives
```

2. `$(date)`
   is a **command substitution** that executes the `date` command and includes its output in the echo statement.
3. `echo "Running program $0 with $# arguments with $$"`
   $0\--> thenameofthescriptthatrunning\# `--> the number of arguments that are giving to the command` $$` --> the process ID of this command that is running
4. `for file in "$@"; do`
   `$@` --> expand to all the arguments passed to the script
5. `grep foobar "$file" > /dev/null 2> /dev/null`
   `grep` --> search for a `pattern` in file,
   e.g. check if the file has `foobar`, if it has --> a zero exit code
   if it doesn't have a `foobar` --> nonzero code

   `> /dev/null` --> redirect the **standard output** and discard
   `2> /dev/null` --> redirect the **standard error** and discard
6. `if [[ "$?" -ne 0 ]]; then` $? `--> the **error code** from the previous command` -ne` --> comparison operator: non equal
7. `echo "# foobar" >> "$file"`
   `>>` --> append at the end of the file
8. `fi` --> mark the end of the `if-block`, `done` --> `for-block`

the running result is as below

1. terminal

```
(base) ishtar@iLouHdeMacBook-Air Desktop % ./example.sh test1.c mcd.sh
Starting program at Fri Feb  9 11:32:03 GMT 2024
Running program ./example.sh with 2 arguments with 26101
File mcd.sh does not have foobar
```

2. in the `mcd.sh` file, the appended comments

Desktop — vim mcd.sh — 80×50

```
mcd (){
    mkdir -p "$1"
    cd "$1"
}
# foobar
```

Debug the shell script

```
shellcheck mcd.sh
```

```
(base) ishtar@iLouHdeMacBook-Air Desktop % shellcheck mcd.sh

In mcd.sh line 1:
mcd (){
^-- SC2148 (error): Tips depend on target shell and yours is unknown. Add a sheb
ang or a 'shell' directive.


In mcd.sh line 3:
    cd "$1"
    ^-----^ SC2164 (warning): Use 'cd ... || exit' or 'cd ... || return' in case
 cd fails.

Did you mean:
    cd "$1" || exit

For more information:
  https://www.shellcheck.net/wiki/SC2148 -- Tips depend on target shell and y...
  https://www.shellcheck.net/wiki/SC2164 -- Use 'cd ... || exit' or 'cd ... |...
```

Find

Find File

1. **find a specific file's path**

```
find . -name src -type d
```

`.` --> stands for **the current folder**

`-name` --> indicated the following file name is `src`

`-type` --> indicated the following type is `d`

```
(base) ishtar@iLouHdeMacBook-Air Desktop % find ./cs61b -name TestUtils.java -type f
./cs61b/proj0/tests/game2048logic/TestUtils.java
```

2. **find multiple paths**

```
find . -path "..." -type f
```

```
(base) ishtar@iLouHdeMacBook-Air Desktop % find "./usa visa" -path '**/LLOYDS/*.pdf' -type f
./usa visa/Bankstatements/LLOYDS/Statement_2023_12.pdf
./usa visa/Bankstatements/LLOYDS/Statement_2023_10.pdf
./usa visa/Bankstatements/LLOYDS/Statement_2023_11.pdf
./usa visa/Bankstatements/LLOYDS/Statement_2024_1.pdf
./usa visa/Bankstatements/LLOYDS/Statement_2024_2.pdf
./usa visa/Bankstatements/LLOYDS/Statement_2023_9.pdf
```

3. find & execute

```
~/m/tools ❯❯❯ find . -name "*.tmp"
./project1/src/test/test2.tmp
./project1/src/test/test3.tmp
./project1/src/test/test1.tmp
./project2/src/test/test2.tmp
./project2/src/test/test3.tmp
./project2/src/test/test1.tmp
~/m/tools ❯❯❯ find . -name "*.tmp" -exec rm {} \;
~/m/tools ❯❯❯ echo $?
0
~/m/tools ❯❯❯ find . -name "*.tmp"
```

Find the content

using `grep`

```
grep foobar -R       # Recursively find all the files which contains "foobar"
```

```
(base) ishtar@iLouHdeMacBook-Air Desktop % grep -R foobar
./example.sh:    grep foobar "$file" > /dev/null 2> /dev/null
./example.sh:        echo "File $file does not have foobar"
./example.sh:        echo "# foobar" >> "$file"
./mcd.sh:# foobar
```

Find the history of CML

```
(base) ishtar@iLouHdeMacBook-Air Desktop % history
 1040  find ./cs61b -name TestUtils -type f
 1041  find ./cs61b -name TestUtils.java -type f
 1042  find ./sc61b -path '**/tests/*.java' -type f
 1043  find ./cs61b -path '**/tests/*.java' -type f
 1044  find "./usa visa" -path '**/LLOYDS/*.pdf' -type -f
 1045  find "./usa visa" -path '**/LLOYDS/*.pdf' -type f
 1046  fd "*.sh"
 1047  find "*.sh"
 1048  find ".*sh"
 1049  locate "usa visa"
 1050  grep foobar test1.c
 1051  grep foo test1.c
 1052  grep -R foobar
 1053  grep -R "inventory management"
 1054  grep -R inventory
 1055  rg "inventory management"
```