SECTION 5

LOGIC

SECTION CHEAT SHEET

CHAINING COMMANDS WITH LIST OPERATORS

KEY DEFINITIONS

LIST	LIST OPERATORS
When you put one or more commands on a given line	Types of control operators that enable us to create lists of commands that operate in different ways

LIST OPERATORS

Operator	Example	Meaning
&	command1 & command2	Sends command1 into a subshell to run "asynchronously" in the background, and continues to process command2 in the current shell.
;	command1 ; command2	Runs command1 and command2, i.e. one after the other. The shell will wait for command1 to complete before starting command2.
&&	command1 && command2	The "and" operator. The shell will only run command2 if command1 is successful (i.e. returns an exit code of 0).
||	command1 || command2	The "or" operator. The shell will only run command2 if command1 is **un**successful (i.e. returns a **non-zero** exit code).

TEST COMMANDS + CONDITIONAL OPERATORS:

TEST COMMANDS

"a command that can be used in bash to compare different pieces of information"

Syntax:

```
[ EXPRESSION ]
```

Operators to use:

OPERATOR	EXAMPLE	MEANING
-eq	[2 -eq 2]	Successful if the two numbers are equal
-ne	[2 -ne 2]	Successful if the two numbers are not equal
=	[$a = $b]	Successful if the two strings are equal
!=	[$a != $b]	Successful if the two strings are not equal
-z	[-z $c]	Successful if a string is empty
-n	[-n $c]	Successful if a string is not empty
-e	[-e /path/to/file]	Successful if a file system entry /path/to/file exists
-f	[-f /path/to/file]	Successful if a file system entry /path/to/file exists **and is a regular file**
-d	[-d /path/to/file]	Successful if a file system entry /path/to/file exists **and is a directory**
-x	[-x /path/to/file]	Successful if a file system entry /path/to/file exists **and is executable by the current user**

IF STATEMENTS:

start and end using the reserved words "if" and "fi"

check the exit status of a command and only runs the command if a certain condition is true

Syntax for if statements:

```
if test1; then
      Commands... # only run if test1 passes
elif test2; then
      Commands... # only run if test1 fails and test2 passes
elif testN; then
      Commands... # only run if all previous tests fail, but testN passes
else
      Commands... # only run if all tests fail
fi
```

Example Script:

```
#!/bin/bash

read -p "Please enter a number" number

if [ $number -gt 0 ]; then
   echo "Your number is greater than 0"
elif [ $number -lt 0 ]; then
   echo "Your number is less than 0"
else
   echo "Your number is 0!"
fi
```

IF STATEMENTS - COMBINING CONDITIONS:

It is possible to chain together multiple test commands using list operators to create more powerful conditions.

Script: If file1.txt equals file2.txt AND file3.txt, then delete file2.txt and file3.txt

```
#!/bin/bash

a=$(cat file1.txt) #  "a" equals contents of file1.txt
b=$(cat file2.txt) # "b" equals contents of file2.txt
c=$(cat file3.txt) #  "c" equals contents of file3.txt

if [ "$a" = "$b" ] && [ "$a" = "$c" ]; then
  rm file2.txt file3.txt
else
  echo "File1.txt did not match both files"
fi
```

Script: If file1.txt equals file2.txt OR file3.txt, then delete file2.txt and file3.txt

```
#!/bin/bash

a=$(cat file1.txt) # "a" equals contents of file1.txt
b=$(cat file2.txt) # "b" equals contents of file2.txt
c=$(cat file3.txt) # "c" equals contents of file3.txt

if [ "$a" = "$b" ] || [ "$a" = "$c" ]; then
  rm file2.txt file3.txt
else
  echo "File1.txt did not match either file"
fi
```

CASE STATEMENTS:

Case statements provide us with an elegant way to implement branching logic, and are often more convenient than creating multiple “elif” statements.

The tradeoff, however, is that case statements can only work with 1 variable.

Case statements start and end using the reserved words “case” and “esac”

Syntax for case statements:

```
case "$variable" in # don't forget the $ and the double quotes!
  pattern1)
    Commands ...
    ;;
  pattern2)
    Commands ...
    ;;
  patternN)
    Commands ...
    ;;
  *)
    Commands ... # run these if no other pattern matches
    ;;
esac
```

Example Script:

```
#!/bin/bash

read -p "Please enter a number: " number

case "$number" in
    "") echo "You didn't enter anything!"
    [0-9]) echo "you have entered a single digit number";;
    [0-9][0-9]) echo "you have entered a two digit number";;
    [0-9][0-9][0-9]) echo "you have entered a three digit number";;
    *) echo "you have entered a number that is more than three digits";;
esac
```

KEY POINTS ON CASE STATEMENTS:

1. It's very important to **remember to use a $ in front of the variable name** otherwise the case statement won't work, as it cannot access the variable's value

2. **Remember to wrap the expansion of the variable name in double quotes** to avoid word splitting issues

3. Patterns follow the **same rules as globbing patterns.**

4. **Patterns are evaluated from top to bottom**, and only the commands associated with the first pattern that matches will be run.

5. ***) is used as a "default" case,** and is used to hold commands that should run if no other cases match.