COMP2101
Summer 2022

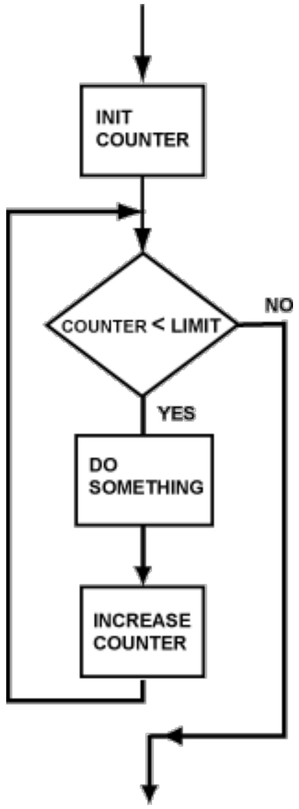
Loops and Arrays

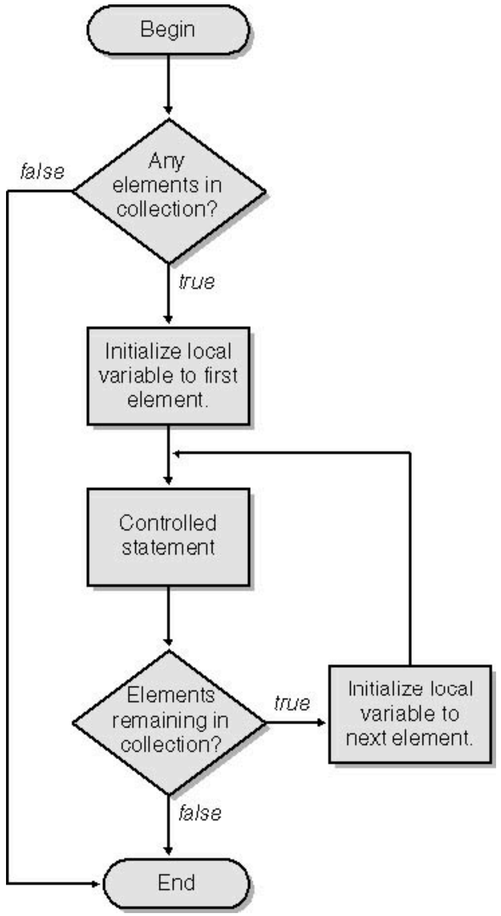
Script Block Looping

- An action list can be executed repeatedly (known as a loop) based on the success or failure of a testlist using the while command
- The break or continue commands can be used in the action list to get out of a loop early
- break jumps to the done command and continues the script past the loop
- continue jumps back to the while command to redo the testing list

```
while testlist; do
   actionlist
done
```

Looping With For





https://stackoverflow.com/questions/20580028/flowchart-for-each-loop-loop-without-variable-increment

- Looping in a script using for
  - Performing a task a set number of times
  - Doing a task with each item in a list of data items

https://chortle.ccsu.edu/assemblytutorial/Chapter-18/ass18_10.html

For Command

- The for command allows repeated execution of a list with each word from a word list
- The for command specifies a variable name
- The first word in the list is put into the variable and the action list is run
- Then the next word in the list is put into the variable and the action list is run, until there are no more words to get from the list

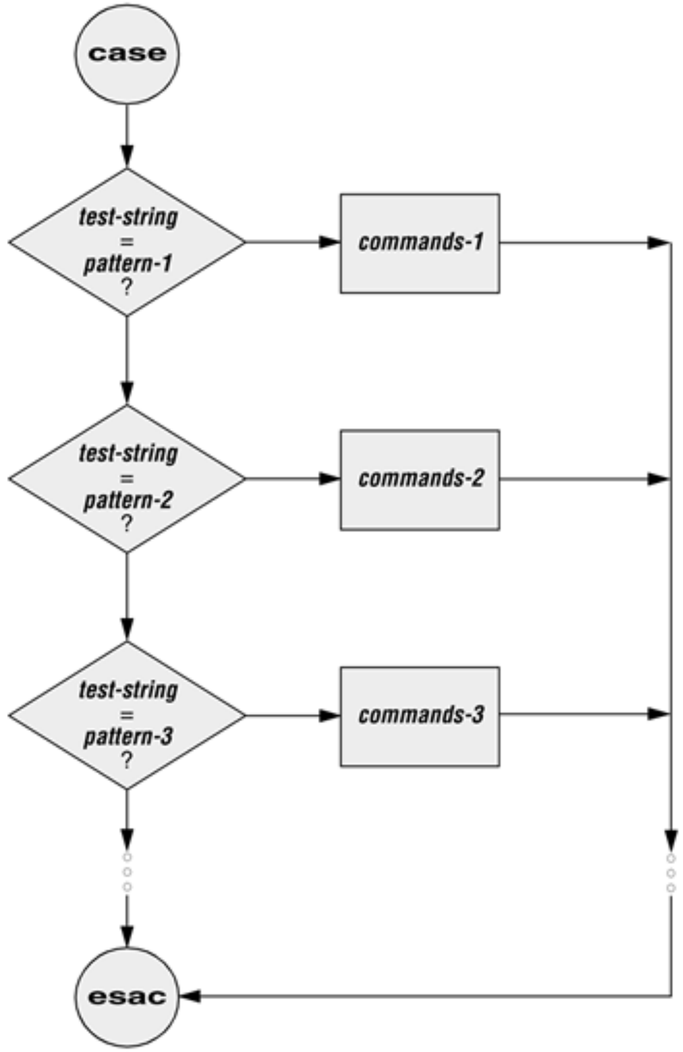
```
for varname in wordlist; do
    list
done
```

For Command

- The for command allows repeated execution of an action list controlled by evaluating 2 commands and a test expression, and is commonly used for loops that use a counter
- The initial command is optional, but when present it is run once
- The test expression is tested before the action list executes
- The loop command is optional, but when present it runs after action list execution
- After the loop command runs, the test is done to see whether to execute the action list and loop command again

```
for (( initial command; test expression; loop command )); do
    list
done
```

Matching Patterns With Case



https://flylib.com/books/en/3.161.1.128/1/

- Multiple reference value tests for a single variable - using case
- Command line arguments - adding options to your scripts

Case Command

- The case command allows execution of a list based on the value of a variable or word
- Very commonly used to process the special variables $1, $2, $3, etc.
- It allows comparing a variable's contents to multiple reference values or patterns

```
case $var in
    pattern )
        list
        ;;
    pattern | pattern )
        list
        ;;
    * )
        list
        ;;
esac
```

Command Line Arguments

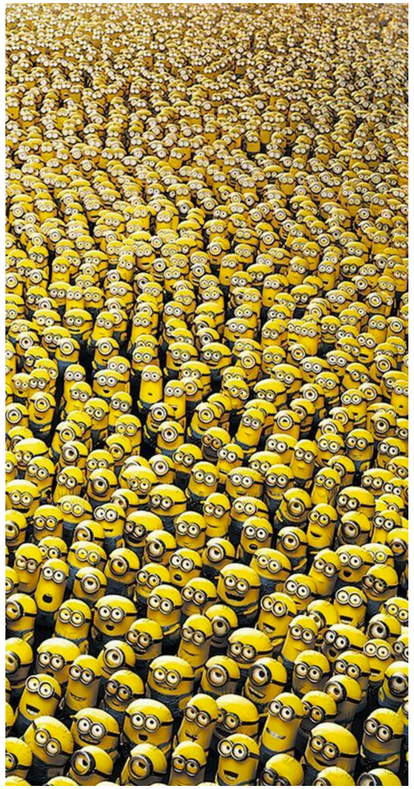
- Any command may have options and arguments
- The command line to run a script is accessible by the script, using the special variables $0, $1, $2, etc.
- $0 holds the command itself and $# holds a count of how many words are on the command line other than the command itself
- $1, $2, $3, etc. hold each of the remaining words from the command line
- On a bash command line, words are space-separated sequences of characters
- Quoting and the escape character can be used to create user-desired word boundaries on the command line (e.g. "My File" becomes a single word)

Command Line Processing

- A loop can be used to cycle through the available command line arguments and interpret what is there
- We can use shift to renumber the command line variables each time through the loop
- Requiring arguments of the form -x or --option-name is known as using named arguments
- The case statement is better than the if statement for this

```
while [ $# -gt 0 ]; do
    case "$1" in
    -h | --help )
        echo "Usage: $(basename $0) [-h|--help]"
        exit
        ;;
    * )
        echo "Unrecognized argument '$1'"
        exit 1
        ;;
    esac
    shift
done
# Command line processed
# Named arguments recognized and saved as needed
```

Arrays



https://i.pinimg.com/originals/7f/be/ff/7fbeffd99fa54411109c6f411cd934a9.jpg

https://www.freecodecamp.org/news/sorting-algorithms-explained-with-examples-in-python-java-and-c/

- Handling collections of data items using arrays
- Debugging scripts

Array Variables

- Arrays allow us to group data items and still operate on them as individual items
- Elements of an array are uniquely identified by an index integer starting at 0
- Elements of an array can be accessed using the varname[index] syntax
- Negative indices count backwards from the end of the array
- @ and * can be used as indices to include all array elements
- # can be used to obtain a count of elements in an array

```
arrayvar=(a b c)
echo ${arrayvar[0]} ${arrayvar[-1]}
echo ${arrayvar[@]} ${#arrayvar[@]}
```

INDEX	DATA
0	a
1	b
2	c

Unnamed Arguments

- Sometimes you need one or more data items for a script and want it on the command line, but don't want the user to have to put option letters or names in front of it (e.g. fixmydir dirname1 dirname2)
- In your command line processing, assign things found on the command line without a leading dash to a variable which stores the list of data items from the command line
- Then you can examine that variable to see what the user gave you to work on

```
declare -a stuffToProcess

while [ $# -gt 0 ]; do
        case "$1" in
        -h | --help )
                echo "Usage: $0 [-h] [stuff ...]"
                exit 0
                ;;
        * )
                stuffToProcess+=("$1")
                ;;
        esac
        shift
done
[ ${#stuffToProcess[@]} ] && echo "Will do work on ${stuffToProcess[@]} (${#stuffToProcess[@]} items)"
```

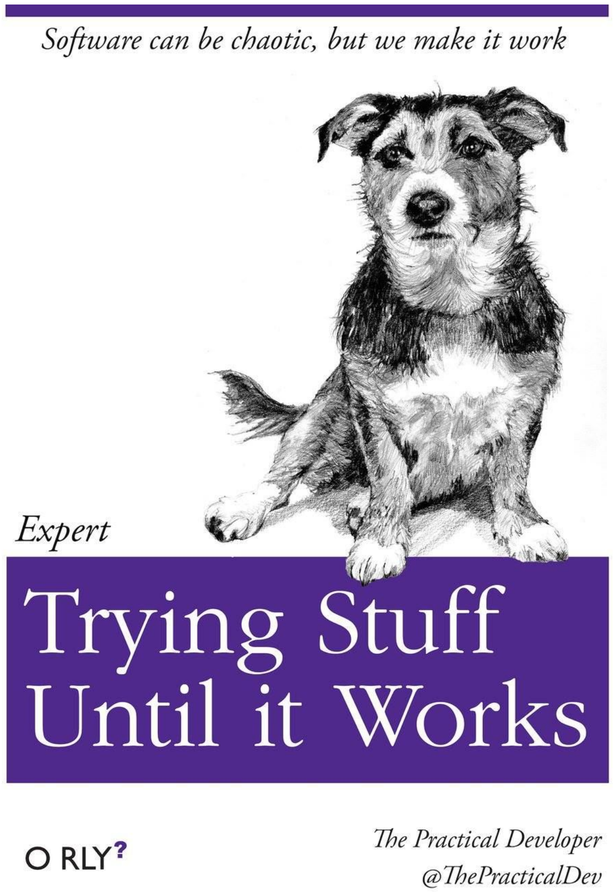
Associative Arrays

- Associative arrays (sometimes called hashes) use a string as an index
- They must be declared before being used
- They are useful for storing structured data

```
declare –A foo
foo=([key1]="data1" [key2]="data2" [key3]="data3")
echo ${foo[key1]}
echo ${foo[@]}
echo ${!foo[@]}
```

INDEX	DATA
key1	data1
key2	data2
key3	data3

Working with bigger data



- arrays and looping
- fourth challenge script