CIS241

System-Level Programming and Utilities

bash Loops and Arrays

Erik Fredericks, frederer@gvsu.edu

Fall 2025

> **Based on material provided by Erin Carrier, Austin Ferguson, and Katherine Bowers**

For loops - similar to Python

```
for var in list; do
   # loopy things
done
```

Samples:

```
for var in 1 2 3 4 5
for var in "alice" "bob" "erik"
for val in {0..10..2}
for var in $@
```

Syntax note

Remember that `$(cmd)` will run `cmd`

```
var=$(ls)

for idx in $(seq 0 10); do
  echo $idx
done
```

While loops

```
while CONDITION; do
   # while things
done
```

`CONDITION`? Same as `if`!

Fun use: parsing arguments

`getopts`

Flags/switches

- E.g., `-f  -r -t`
  Option argument
- E.g., `-d /path/to/dir`
  Other arguments
- E.g., `"this is a string being passed in as a single argument"`
  Example:
- `sh script.sh –f –r –t –d /path/to/dir "this is that string I mentioned"`

Parsing arguments

`OPTSTRING`

- Define legal argument options

`VARNAME`

- Which variable to store argument data within

`getopts OPTSTRING VARNAME`

`getopts abc:d`

Allows:

- `-a` `-b` `-c <SOME DATA>` `-d`

`getopts :abc:d`

- Suppress error output (first colon)

Parsing arguments with `getopts`

Parses arguments in order

- Looping over `getopts` (calling multiple times) common

```
while getopts :ci opt; do
  case $opt in
    c)
      echo “-c found” ;;
    i)
      echo “-i found” ;;
    \?)
      echo “invalid option -$OPTARG” >&2 ;;
  esac
done
```

Add colon (`:`) after flag to indicate an expected value

- E.g., `t`:

```
while getopts :csi: opt; do
  case $opt in
    c)
      echo “-c found” ;;
    s)
      echo “-s found” ;;
    i)
      echo “-i found with argument $OPTARG” ;;
    \?)
      echo “invalid option -$OPTARG” >&2 ;;
  esac
done
```

Reading user input

User input : read from command line

```
echo –n "Enter your name: "    # -n skips the newline
read your_name
echo "Hello $your_name"
```

`read` by default dumps everything into single variable (`$your_name` in this case)

```
read –p "Enter your name: " first_name last_name
echo “Your name is $first_name $last_name
```

Timeout (or, not waiting for those pesky users)

May wish to provide a way to give a default response if the user isn't watching

```
read -t 5 -p "Should we auto-continue [Y/n]? " answer
case $answer in
  N | n)
    echo "Exiting..."
    exit
    ;;
  Y | y | *)
    echo
    echo "Continuing on..."
    ;;
esac
```

Arrays

Creating arrays

```
arr=()              # Empty array
declare -a arr_2 # Also empty
arr_3=(10 8 6)     # Create with values
```

Setting/Adding values

```
arr[0]=test    # Set value
arr_2+=(87)    # Append value
arr_3+=(4 2) # Append values
```

Accessing arrays

Accessing:

```
${arr[0]}  # Access nth element
```

Special accessing:

```
${arr[@]}                  # All elements
${arr[*]}                  # All elements as string
${#arr[@]}                 # Count of elements
${!arr[@]}                 # Indices of elements
${arr[@]:start:num}  # Slice of array
```

Iterating over arrays (hopefully a reminder)

```
for x in ${arr[@]}; do
  echo "$x"
done
```

Associative arrays (maps/dictionaries)

```
declare -A dnd_stats
dnd_stats=(
  [str]=16
  [dex]=10
  [con]=18
  [int]=11
  [wis]=10
  [cha]=14
)
echo "wisdom = ${dnd_stats[wis]}"
dnd_stats[wis]=$((dnd_stats[wis] + 2))
echo "wisdom = ${dnd_stats[wis]}"
```