CS 35L: Shell Scripting

Lab 6
Hengda Shi
Week 2 Lecture 2

Regular Expression Task 1

- Matching username
  - 3 to 16 characters in length
  - Can have lowercase letters, digits, underscore (_), hyphen (-)
- ANS: ^[a-z0-9_-]{3,16}$

Regular Expression Task 2

{0^i1^j | i is even and j is odd }

ANS: ^(00)*1(11)*$

Regular Expression Task 3

Binary strings that end with 1 and do not contain the substring 00

ANS: ^(1|01)+$

Regular Expression Task 4

Odd-length binary strings that begin and end with the same symbol

ANS: ^([01]|0[01]([01][01])*0|1[01]([01][01])*1)$

Regular Expression Task 5

Write an extended regular expression (ERE) to match any line that is exactly an integer with an optional leading + or - sign. By "exactly" we mean that the regex has ^ and $ at the two ends. Except for the optional sign, the integer is a non-empty consecutive sequence of digits that cannot start with a 0 except when the integer is 0.

e.g. **should** match these:

0

9

10

+0

+23

-0

-100

and **should not** match these:

000

023

+

-

+02

-001

+-2

-+4

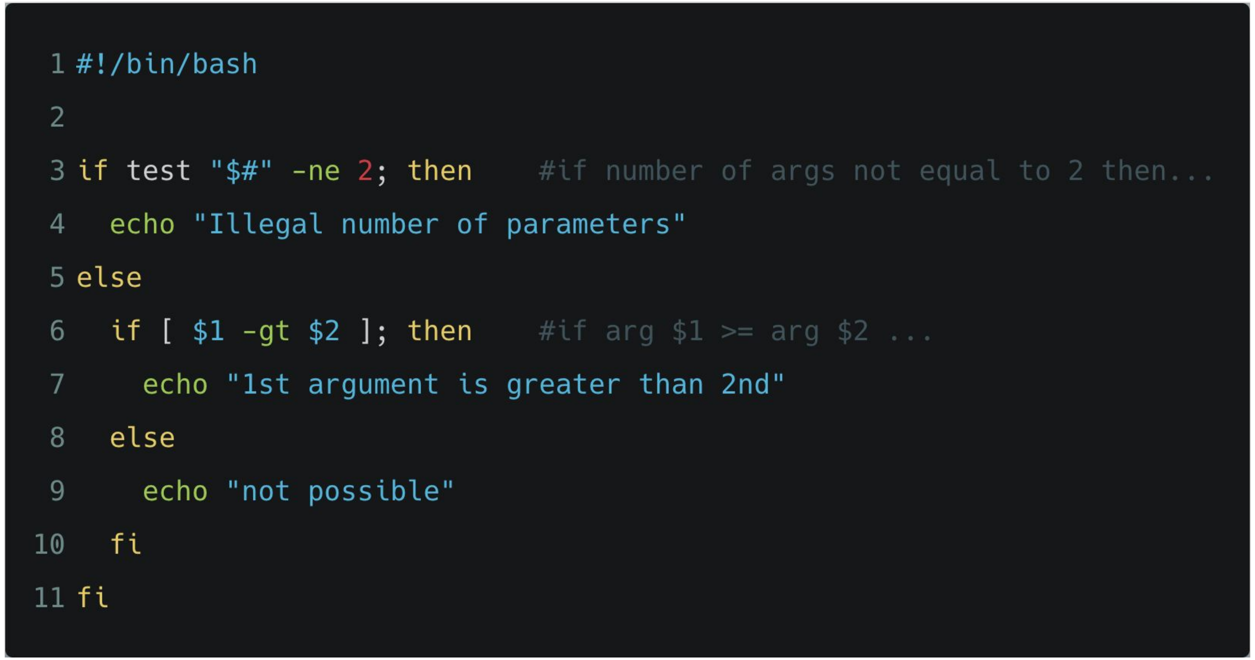
ANS: ^[-+]?(0|[1-9][0-9]*)$

if Statement

```
if condition
then
        statements-if-true-1
elif condition
then
        statements-if-true-2
else
        statements-if-all-else-fails
fi
```

if Statement (cont'd)

- If statement use "test" command or "[" command as a statement.
- `$ man test` for all kinds of condition you could use



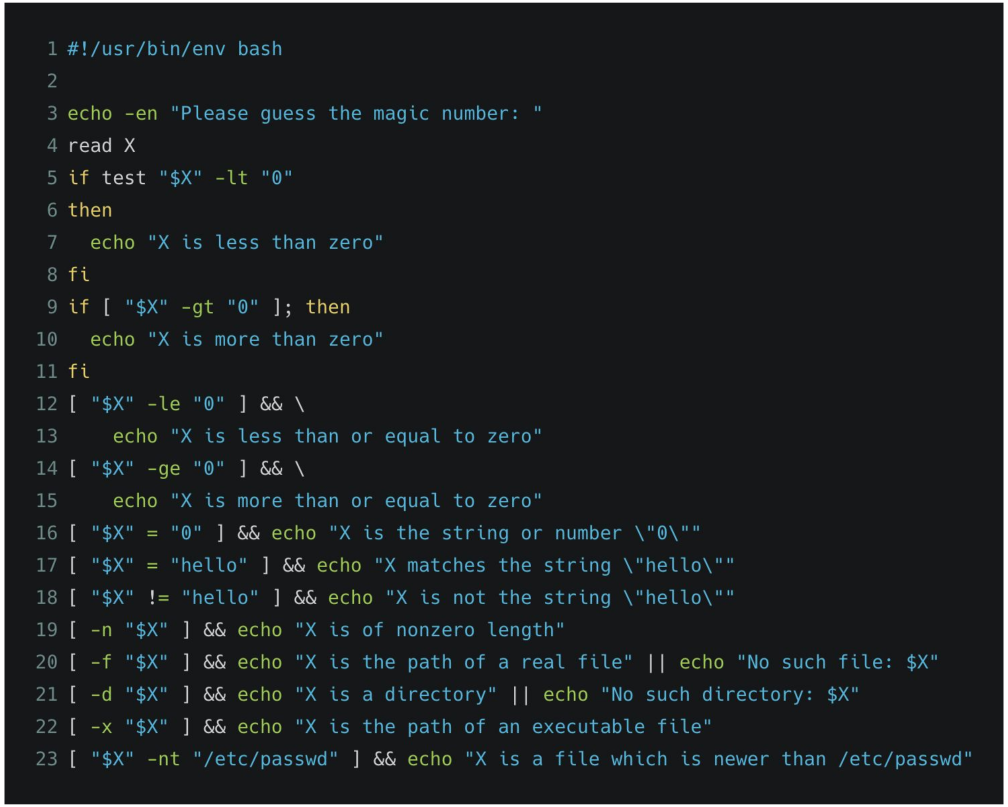
```
#!/bin/bash

if test "$#" -ne 2; then    #if number of args not equal to 2 then...
  echo "Illegal number of parameters"
else
  if [ $1 -gt $2 ]; then    #if arg $1 >= arg $2 ...
    echo "1st argument is greater than 2nd"
  else
    echo "not possible"
  fi
fi
```

test

- test program is often used in if statement conditions
- [is also a program that is equivalent to test
- That means [must be surrounded by spaces
  - if [$foo = "bar"] will not work
  - if [$foo = "bar"] will work
- Use: -eq, -lt, -gt, … for integers
- Use =, >, <, !=, … for strings
- Test if a file descriptor is open on terminal
  - $ test -t 2 && echo "YES" || echo "NO"

test (cont’d)



```bash
#!/usr/bin/env bash

echo -en "Please guess the magic number: "
read X
if test "$X" -lt "0"
then
  echo "X is less than zero"
fi
if [ "$X" -gt "0" ]; then
  echo "X is more than zero"
fi
[ "$X" -le "0" ] && \
    echo "X is less than or equal to zero"
[ "$X" -ge "0" ] && \
    echo "X is more than or equal to zero"
[ "$X" = "0" ] && echo "X is the string or number \"0\""
[ "$X" = "hello" ] && echo "X matches the string \"hello\""
[ "$X" != "hello" ] && echo "X is not the string \"hello\""
[ -n "$X" ] && echo "X is of nonzero length"
[ -f "$X" ] && echo "X is the path of a real file" || echo "No such file: $X"
[ -d "$X" ] && echo "X is a directory" || echo "No such directory: $X"
[ -x "$X" ] && echo "X is the path of an executable file"
[ "$X" -nt "/etc/passwd" ] && echo "X is a file which is newer than /etc/passwd"
```

ref: https://www.shellscript.sh/test.html

for loop

```
for i in file1 file2 file3
do
        statements
done
-----------------------------------------------------------------
for i in {0..5}             # only with bash v3.0+
do
        statements
done
-----------------------------------------------------------------
for i in {0..5..2}          # only with bash v4.0+
do
        statements
done
```

for loop (cont’d)

```
nums=('0' '1' '2' '3')
for i in "${nums[@]}"
do
        echo "num: $i"
done
--------------------------------------------------------------------
for ((i = 0 ; i <= 1000 ; i++))
do
  echo "Counter: $i"
done
```

while loop

Basic Syntax:

```
    while condition
    do
        statements
    done
```

Infinite loop:

```
    while :                         # or while true
    do
        statements
    done
```

while loop (cont’d)

Read a file line by line:

```
file=/etc/passwd
while read -r line; do
  echo $line
done < "$file"
```

Case statement

```
case EXPRESSION in
  PATTERN_1)
    STATEMENTS
    ;;
  PATTERN_2)
    STATEMENTS
    ;;
  PATTERN_N)
    STATEMENTS
    ;;
  *)
    STATEMENTS
    ;;
esac
```

break and continue

- They work similarly as in C/C++

Function

- Semantically, calling a function is very similar to invoking another bash script
- Must be defined before they can be used
- Can be done either at the top of a script or by having them in a separate file and source them with the “dot” (.) command

Function (cont’d)

Basic Syntax:

```
func_name () {
  statements
}
----------------------------------------
function function_name {
  statements
}
```

Calling a function:

```
func_name arg1 arg2         # function call
```

Scope of Variables

- Default scope of variable is global, other than the parameters ($1, $2, $@, etc).
- But you can use the local keyword to limit the scope of variable

```
#!/usr/bin/env bash
myfunc ()
{
  local lvar="Local Content"
  echo "I was called as : $@"
  x=2
  echo "Inside function call: ${lvar}"
}
echo "Script was called with $@"
x=1
echo "x is $x"
echo "Before function call: ${lvar}"
myfunc 1 2 3
echo "After function call: ${lvar}"
echo "x is $x"
```

Function return

- The return command serves the same function as exit and works the same way:

```
answer_the_question ( ) {
        ...
        return 42
}
```

Simple Execution Tracing

- Let shell prints out each command as it executes
- Turn on execution tracing in script by
  - set -x   # turn on
  - set +x   # turn off

IFS (Internal Field Separator)

- This variable determines how Bash recognizes fields, or word boundaries, when it interprets character strings.
- $IFS defaults to whitespace (space, tab, and newline), but may be changed
- echo "$IFS" (With $IFS set to default, a blank line displays)
- More details: http://tldp.org/LDP/abs/html/internalvariables.html
- Example:

```
cell='123-456-7890'
# IFS tells which character should split words. Defaults to whitespace
old_IFS="$IFS"
IFS="-"
for num in $cell; do
  echo “${num}”
done
IFS=${old_IFS}
```

Lab Hints

- The point of the lab is to practice with tr, grep and sed commands
- It is not enforced to delete all English words, but the majority of it that do not follow Hawaiian rule should be gone
- tr vs. sed: tr works on characters; sed works on string/lines
- Don’t forget to escape special characters
- Treat “aaa-bbb” as two words: “aaa” and “bbb”
- Please make sure your script will at least run, otherwise we cannot give you any points
- More hints on piazza!

Homework Hints

- Basic poornames
  - Use find or ls combined with regex to parse
  - Valid characters are only: A-Za-z._
  - “No two files in the same directory can have names only differ in case”
- Recursive poornames
  - It’s essentially a search problem on top of basic poornames
  - One direction is check all subdirectories in DFS style
  - Potential solutions could use:
    - Commands exec or sh for recursive calls
    - find -maxdepth -mindepth to restrict recursion
    - uniq for cleanup