CSI3660 – System Administration

Prof. Fredericks

Bash Scripting Pt. 2

Outline

- Lab walkthrough
- Decision statements
- Loops

Lab – Script 1: Output & Args

- 3 command line parameters (arguments)
- Run as:
  - sh <lastname>_Script1.sh <first name> <username> <directory list command>
- **Must** use command line argument to list directory
- Output:
  - Hello <first name>, the contents of my home directory are:
  - <Listing of home directory>

Lab – Script 2: Loop & Modulo

- Execute for loop
  - Loop over range of 1-15
  - Calculate and output modulo 2 of each loop iteration value
  - Output should be:
    - 1 % 2 = 0
    - 2 % 2 = 1
    - 3 % 2 = 0
    - …

Lab – Script 3: Automate User Tasks

- Create an array of at least 5 users
- Print each name to the console
- Add each user to system
  - Show that users were added
- Delete each user, including home directory, from system
- Demonstrate users and directories are gone

Decision Constructs

- if statement
- case statement

Decision Constructs

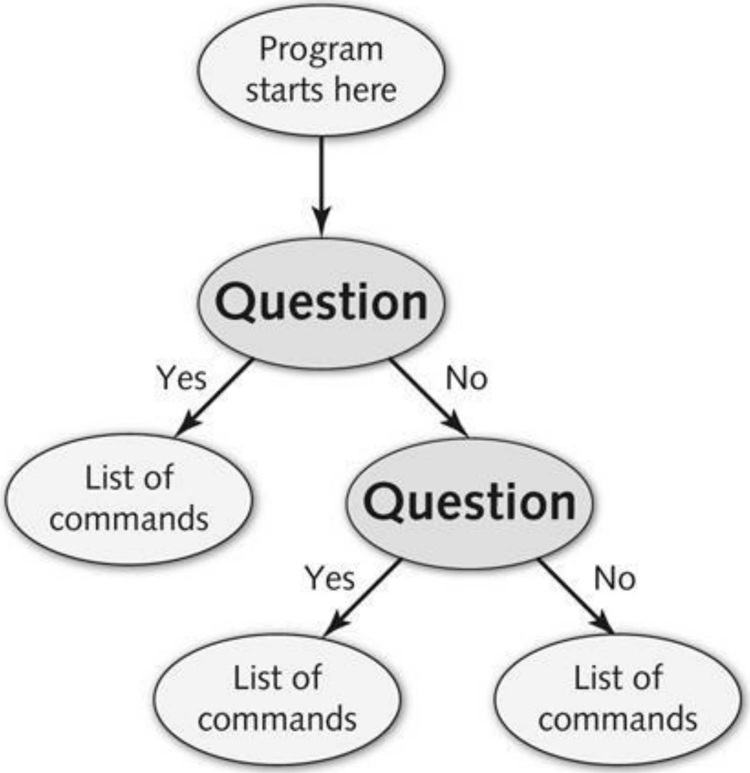
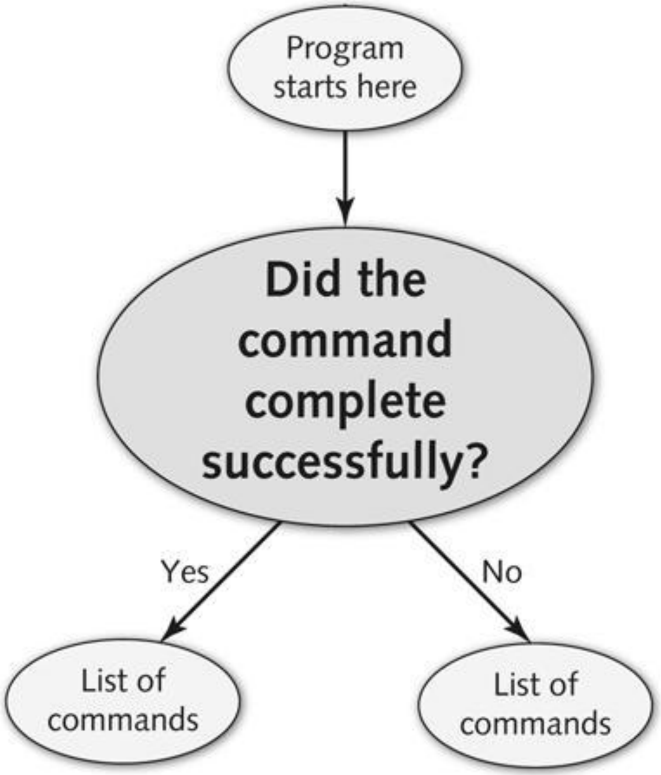


Figure 7-4: A two-question decision construct

Decision Constructs



Exit status: 0

if construct

Spaces between brackets are REQUIRED

- "If statement"

In C:

```
if (cond)
{
  // statements
}
```

Shorthand for **test** command (p304)

```
char myvar = 'a';
if (myvar == 'a')
{
  printf("this was true");
}
```

In bash:

```
if [ cond ]
then
  # statements
fi

OR

if [ cond ]; then
  # statements
fi
```

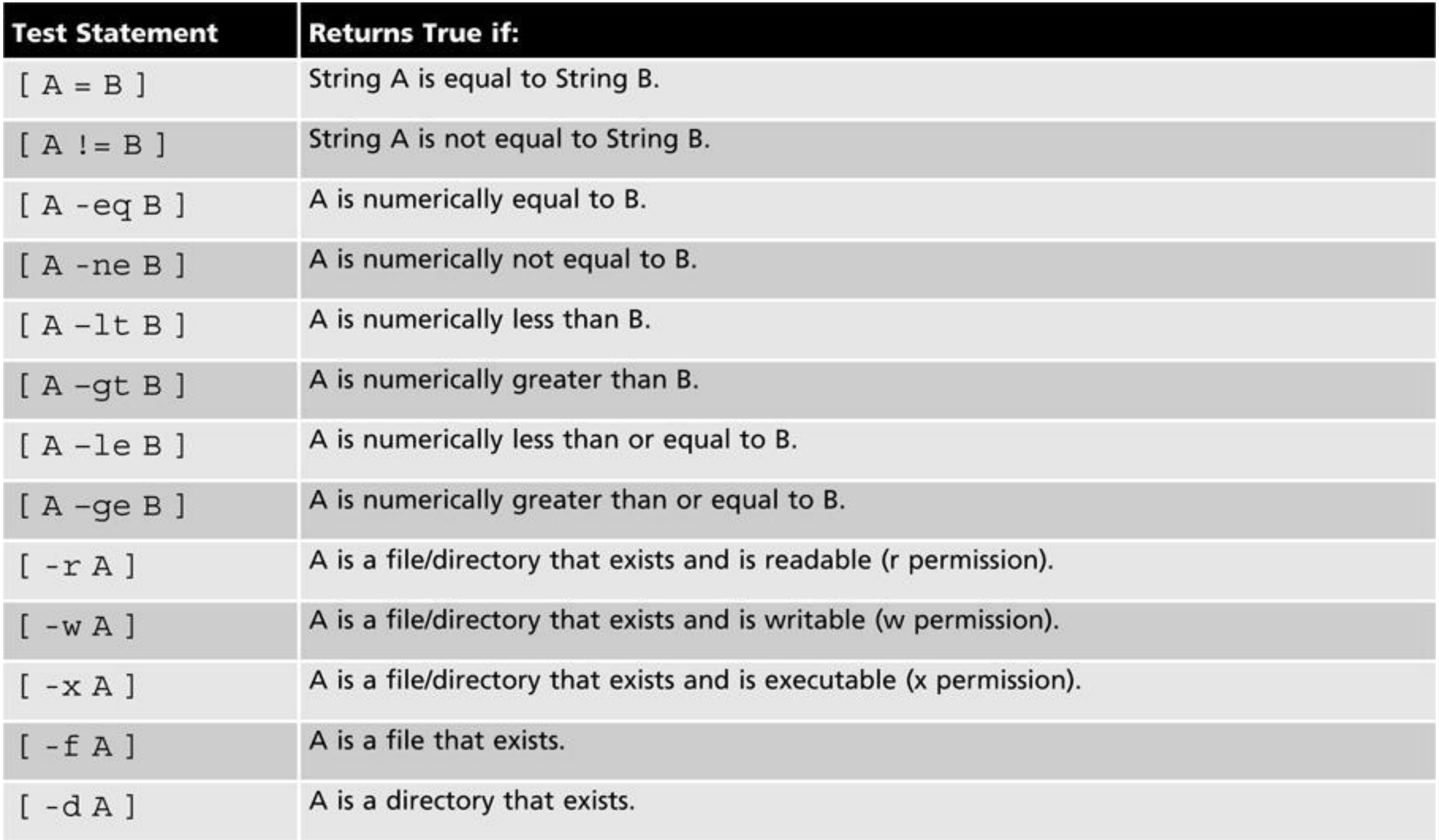
```
myvar="a"
if [ $myvar = "a" ]; then
  echo "this was true"
fi
```

Checking Command Output

```
if [ `pwd` ]; then
   echo 'success'
else
   echo 'failure'
fi
```

```
echo $? → EXIT STATUS
```

Common Conditional Tests



Test Statement	Returns True if:
`[ A = B ]`	String A is equal to String B.
`[ A != B ]`	String A is not equal to String B.
`[ A -eq B ]`	A is numerically equal to B.
`[ A -ne B ]`	A is numerically not equal to B.
`[ A –lt B ]`	A is numerically less than B.
`[ A –gt B ]`	A is numerically greater than B.
`[ A –le B ]`	A is numerically less than or equal to B.
`[ A –ge B ]`	A is numerically greater than or equal to B.
`[ -r A ]`	A is a file/directory that exists and is readable (r permission).
`[ -w A ]`	A is a file/directory that exists and is writable (w permission).
`[ -x A ]`	A is a file/directory that exists and is executable (x permission).
`[ -f A ]`	A is a file that exists.
`[ -d A ]`	A is a directory that exists.

if…elif…else

- Multiple actions for if statement conditions

```
myvar="b"
if [ $myvar = "a" ]; then
  echo "this was true"
else
  echo "this was false"
fi
```

```
myvar="c"
if [ $myvar = "a" ]; then
  echo "this was true"
elif [ $myvar = "b"]; then
  echo "myvar was actually b\!"  # need to escape char
else
  echo "turns out it wasn't a or b"
  exit 1   # exit with status code 1
           # exit by itself means exit 0 → success
fi
```

Nested if statements

```
myvar="5"
if [ $myvar = "a" ]; then
  echo "this was true"
elif [ $myvar = "b" ]; then
  echo "myvar was actually b\!"  # need to escape char
else
  echo "turns out it wasn't a or b"

  if [ $myvar = "c" ]; then
    echo "turns out it was c"
    exit   # exit successfully
  fi

  exit 2   # exit with status code 2 - failure
fi
```

Adding Extra Conditions

```
myvar=22

if [ $myvar -eq "20" -o $myvar -gt "21" ]; then
   echo "myvar is 20 or greater than 21"
   exit   # exit successfully
fi
```

However

We should use the Bash-style syntax

```
if [[ $var1 == "hi" ]]; then
  echo "hello there"
fi
```

Why?

If var1 doesn't exist, the old style doesn't like it

Try the above with prior style

```
if [ $var1 = "hi" ]; then
  echo "hello"
fi
```

vs.

```
if [ "$var1" == "hi" ]; then
  echo "hello"
fi
```

Alternate Syntax

```
myvar=5
if (( $myvar == 5 )); then
  echo "myvar is 5"
fi

# OR

if (( $myvar >= 3 )); then
  echo "myvar is 3 or higher"
fi

# OR

if (( $myvar < 2 )) || (( $myvar >= 3 )); then
  echo "less than 2 or greater than or equal to 3"
fi
```

Double parentheses – expand math expression

Test command

- Check if variable exists or has content

```
myvar=2

if test $myvar
then
   echo 'myvar has data'
fi

#OR

if test $myvar2
then
  echo 'myvar2 has data'
else
  echo 'myvar2 has no data'
fi
```

Less Than / Greater Than Caveats

- What is wrong with:

```
test1=4
test2=10
if [ $test1 > test2 ]; then
  echo 'success'
else
  echo 'fail'
fi
```

```
if [ $test1 \> test2 ]; then
```

Case Statements

- Test single variable for multiple values in a more concise manner than multiple **elif** statements

```
test1="HELLO"
if [ $test1 == "HI" ]; then
  # do something
elif [ $test1 == "ANOTHER STRING" ]; then
  # do something else
# ……and on and on
```

Case Statements

```
test1="HELLO"

case $test1 in
HELLO | HI)
  echo "case successful: $test1" ;;
ANOTHER)
  echo "a different case" ;;
"AND ANOTHER")
  echo "different yet again";;
*)
  echo "default case" ;;
esac
```

Loops

- for loop
- foreach loop
- while loop

for loop

```
# iterate over list
for val in Val1 Val2 Val3; do
  echo $val
done
```

```
# iterate over range
for i in {1..10}; do
  echo $i
done
```

```
# C-style for loop
for (( i=0; i<50; i++ )); do
  echo $i
done
```

foreach loop (iterate over list)

```
#iterate over list
list1="string1 string2 string3"
for val in $list1; do
  echo $val
done


#iterate over array
arr=(one two three four)
for val in "${arr[@]}"; do
  echo $val
done
```

while and until

- Similar format to **for** loop

```
# while loop
while <test>; do
  #commands
done

# until loop
until <test>; do
  #commands
done
```

Difference?

while: break when condition is false

until: break when condition is true

While vs Until

```
i=0
while (( $i == 0 )); do
  echo $i
  i=$((i+1))
done
```

```
i=20
until (( $i == 0 )); do
  echo $i
  i=$((i-1))
done
```

Break and Continue

- Control operation of loop
- **break**
  - Stop execution of **current** (innermost) loop early

```
i=0
while (( $i < 10 )); do
  echo $i
  i=$((i+1))

  if [ $i -eq 5 ]; then
    break
  fi
done
```

```
i=0
while (( $i < 10 )); do
  echo $i
  i=$((i+1))

  for j in {1..50}; do
    if [ $i -eq $j ]; then
      break
    fi
  done

done
```

Continue

- Similar to **break**
- Rather than stopping loop completely, it stops processing the current iteration and continues on with the loop

A Practical Example

- Let's back up the .bashrc and .bash_profile files for each account
- Let's also schedule it to run weekly

Two Parts

- Script File
  - Demo
- Cron job
  - Place script in /etc/cron.weekly
- Or run right now!
  - `$ at -f backup_script.sh now`

Additional Command Line Arg Info

- Need to know the name of the script?
  - `$0`
- What happens if there are more than 9 command line arguments?
  - Reference as: `${10}`
    - Disambiguate from $1..

Testing Cmd Line Args

- You should test for existence of argument if you require its use!
  - Script breaks down if data doesn't exist…

```
if [ -n "$1" ]; then
  echo "Argument $1 exists, continue on…"
else
  echo "Argument missing!  Bail out…"
fi
```

Testing Cmd Line Args

- Check number of arguments
  - If you are expecting 3 and there only is 2, then there may be a problem!

```
$# → number of arguments



if (( $# != 3 )); then
  echo "Incorrect number of arguments!"
  exit 2
fi
```

Shifting Arguments

- **shift**
  - Move argument position
  - $3 → $2
  - $2 → $1
  - $1 → discarded!
- Why shift arguments?

Shifting Arguments

```
cnt=1

while test ${#} -gt 0 # test param count
do
    echo "Parameter $cnt = $1"
    cnt=$[ $cnt+1 ]
    shift
done
```