# Shell Control Structures

CSE 2031
Fall 2011



## Control Structures

- if then else
- for
- while
- case (which)
- until

## if Statement and test Command

- Syntax:

```
if condition
then
   command(s)
elif condition_2
then
   command(s)
else
   command(s)
fi
```

- Command **test** is often used in ***condition***.

3

## **if – then – else** Example

```
% cat if_else
#!/bin/sh
echo -n 'Enter string 1: '
read string1
echo -n 'Enter string 2: '
read string2
if test $string1 = $string2
then
        echo 'They match!'
else
        echo 'No match!'
fi
```

```
% if_else
Enter string 1: acd
Enter string 2: 123
No match!

% if_else
Enter string 1: 123
Enter string 2: 123
They match!
```

4

# `test` Command

| Argument | Test is true if . . . |
|---|---|
| `-d` *file* | *file* is a directory |
| `-f` *file* | *file* is an ordinary file |
| `-r` *file* | *file* is readable |
| `-s` *file* | *file* size is greater than zero |
| `-w` *file* | *file* is writable |
| `-x` *file* | *file* is executable |
| `!` `-d` *file* | *file* is not a directory |
| `!` `-f` *file* | *file* is not an ordinary file |
| `!` `-r` *file* | *file* is not readable |
| `!` `-s` *file* | *file* size is not greater than zero |
| `!` `-w` *file* | *file* is not writable |
| `!` `-x` *file* | *file* is not executable |
| `-e` *file* | *file* or directory exists |

5

# `test` Command (2)

| | |
|---|---|
| *n1* `-eq` *n2* | integer *n1* equals integer *n2* |
| *n1* `-ge` *n2* | integer *n1* is greater than or equal to integer *n2* |
| *n1* `-gt` *n2* | integer *n1* is greater than integer *n2* |
| *n1* `-le` *n2* | integer *n1* is less than or equal to integer *n2* |
| *n1* `-ne` *n2* | integer *n1* is not equal to integer *n2* |
| *n1* `-lt` *n2* | integer *n1* is less than integer *n2* |
| *s1* `=` *s2* | string *s1* equals string *s2* |
| *s1* `!=` *s2* | string *s1* is not equal to string *s2* |

- Parentheses can be used for grouping test conditions.

6

## `test` Example 1

```
% cat check_file
#!/bin/sh
if test ! -s $1
then
   echo "File $1 is empty.“
   exit 1
else
   ls -l $1
fi
```

```
% touch z.txt

% check_file z.txt
File z.txt is empty.
```

7

## `test` Example 2

```
% cat check_file
#!/bin/sh
if test $# -eq 0
then
   echo Usage: check_file file_name
   exit 1
fi
if test ! -s $1
then
   echo "File $1 is empty.“
   exit 1
else
   ls -l $1
fi
```

8

## `test` Example 3

- What is wrong with the following script?

```
% cat chkex2
#!/bin/sh
# Check if a file is executable.
if test -x $1
then
   echo File $1 is executable.
else
   echo File $1 is not executable.
fi
```

9

## `test` and Logical Operators

- `!`, `||` and `&&` as in C
- Following is better version of `test` Example 3

```
%cat chkex
#!/bin/sh
if test -e $1 && test -x $1
then
   echo File $1 is executable.
elif test ! -e $1
then
   echo File $1 does not exist.
else
   echo File $1 is not executable.
fi
```

10

## `for` Loops

```
for variable in list
do
   command(s)
done
```

- ***variable*** is a user-defined variable.
- ***list*** is a sequence of strings separated by spaces.

11

## `for` Loop Example 1

```
% cat fingr
#!/bin/sh
for name in $*
do
   finger $name
done
```

- Recall that **$*** stands for all command line arguments the user enters.

12

## `for` Loop Example 2

```
% cat fsize
#!/bin/sh
for i in $*
do
   echo "File $i: size `wc -c $i`"
done

% fsize chex chfile chfile2
File chex: size 86 chex
File chfile: size 90 chfile
File chfile2: size 163 chfile2
```

13

## `for` Loop Example 3

```
% cat prdir
#!/bin/sh
# Display all c files in a directory
# specified by argument 1.
#
for i in $1/*.c
do
   echo "======= $i ======"
   more $i
done
```

14

# Arithmetic Operations Using **expr**

- The shell is not intended for numerical work (use Java, C, or Perl instead).
- However, **expr** utility may be used for *simple* arithmetic operations on integers.
- **expr** is not a shell command but rather a UNIX utility.
- To use **expr** in a shell script, enclose the expression with backquotes.
- Example:

```
#!/bin/sh
sum=`expr $1 + $2`
echo $sum
```

- Note: spaces are required around the operator **+** (but not allowed around the equal sign).

15

# **expr** Example

```
% cat cntx
#!/bin/sh
# Count the number of executable files in …
# the current working directory
count=0
for i in *        # what if we replace * with $* ?
do
   if test -x $i
   then
      count=`expr $count  + 1`
      ls -l $i
   fi
done
echo "$count executable files."
```

16

## `while` Loops

```
while condition
do
    command(s)
done
```

- Command **`test`** is often used in ***`condition`***.
- Execute ***`command(s)`*** when ***`condition`*** is met.

17

## `while` Loop Example

```
#!/bin/sh
# Display the command line arguments, one per line.
count=1
argc=$#
while test $count -le $argc
do
   echo "Argument $count is: $1"
   count=`expr $count  + 1`
   shift          # shift arg 2 into arg 1 position
done

# What happens if the while statement is as follows?
# while test $count -le $#
```

18

## **until** Loops

```
until condition
do
    command(s)
done
```

- Command **test** is often used in ***condition***.
- Exit loop when ***condition*** is met.

19

## **until** Loop Example

```
% cat grocery
#!/bin/sh
# Enter a grocery list and …
# store in a file indicated by $1
#
echo To end list, enter \"all\".
item=nothing
until test $item = “all”
do
        echo -n "Enter grocery item: "
        read item
        echo $item >> $1
done
```

20

## `until` Loop Example Output

```
% grocery glist
To end list, enter "all".
Enter grocery item: milk
Enter grocery item: eggs
Enter grocery item: lettuce
Enter grocery item: all
```

```
% cat glist
milk
eggs
lettuce
all
```

21

## `case` Statement

```
case variable in
pattern1) command(s);;
pattern2) command(s);;
. . .
patternN) command(s);;
*)        command(s);;  # all other cases
esac
```

- Why the double semicolons?

22

## `case` Statement Example

```
#!/bin/sh
# Course schedule
echo -n "Enter the day (mon, tue, wed, thu, fri): "
read day
case $day in
   mon)     echo 'CSE2031 2:30-4:30 CLH-H'
            echo 'CSE2021 17:30-19:00 TEL-0016';;
   tue | thu)
            echo 'CSE2011 17:30-19:00 SLH-E';;
   wed)     echo 'No class today.  Hooray!';;
   fri)     echo 'CSE2031 2:30-4:30 LAB 1006';;
   *)       echo 'Day off. Hooray!';;
esac
```

23

## Process-Related Variables

- Variable **$$** is PID of the shell.

```
% cat shpid
#!/bin/sh
ps
echo PID of shell is = $$

% shpid
  PID  TTY          TIME CMD
 5658 pts/75   00:00:00 shpid
 5659 pts/75   00:00:00 ps
11231 pts/75   00:00:00 tcsh
PID of shell is = 5658
```

24

## Process Exit Status

- All processes return exit status (return code).
- Exit status tells us whether the last command was successful or not.
- Stored in variable **$?**
- 0 (zero) means command executed successfully.
- 0 is good; non-zero is bad.
- Good practice: Specify your own exit status in a shell script using **`exit`** command.
  - default value is 0 (if no exit code is given).

25

## Process Exit Status: Example

- An improved version of grep.

```
% cat igrep
#!/bin/sh
# Arg 1: search pattern
# Arg 2: file to search
#
grep $1 $2
if test  $? -ne 0
then
  echo Pattern not found.
fi
```

```
% igrep echo phone
echo –n “Enter name: ”

% igrep echo2 chex
Pattern not found.
```

26

## Next time …

- Shell scripting – part 3
- Reading for this lecture:
  - 3.6 to 3.8 and Chapter 5, UNIX textbook
  - Posted notes (chapter 33)

27