Conditional Statements

- Conditional statements are used to *test* something
  - In *Java* or *C*, they test whether a *Boolean variable* is *true* or *false*
  - In a *Bourne shell script*, the only thing you can test is whether *a command is "**successful**" or "**not successful**"*
- Every *well-behaved* command returns back a return code
  - 0 if it was successful
  - Non-zero if it was unsuccessful (actually a value from 1 to 255)

if Statement

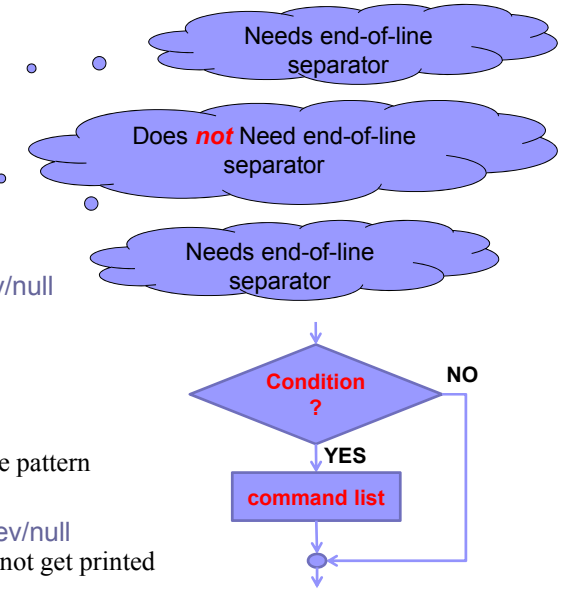
- Simple form:

```
if decision_command
then
   command_set
fi
```

- Example:

```
#!/bin/sh
if  grep "UNIX" myfile >/dev/null
then
   echo "It's there"
fi
```

- grep returns
  - 0 if it successfully finds the pattern
  - non-zero otherwise
- grep output is redirected to /dev/null so that *intermediate* results do not get printed



if and else

- Second form:

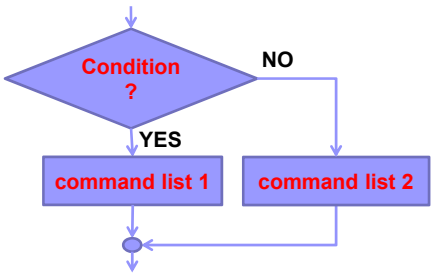
```
if decision_command
then
   command_set_1
else
   command_set_2
fi
```

Does *not* Need end-of-line separator

Does *not* Need end-of-line separator

- Example:

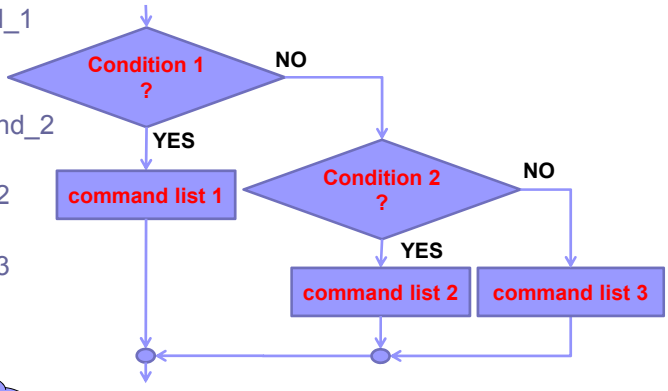
```
#!/bin/sh
if grep "UNIX" myfile >/dev/null
then
  echo  UNIX occurs in myfile
else
  echo  UNIX does not occur in myfile
fi
```



if and elif

- Third form:

```
if decision_command_1
then
   command_set_1
elif decision_command_2
   then
      command_set_2
   else
      command_set_3
fi
```



Only ONE *fi*

if and elif

- Example:

```
#!/bin/sh
if grep "UNIX" myfile >/dev/null
then
   echo "UNIX occurs in file"
elif grep "DOS" myfile >/dev/null
   then
     echo "UNIX does not occur, but DOS does"
   else
     echo "Nobody is there“
fi
```


Use of Semicolons

- Instead of being on separate lines, statements can be separated by a semicolon (;)
- Example 1

```
if grep "UNIX" myfile > /dev/null ; then echo "Got it " ; fi
```

- Example 2

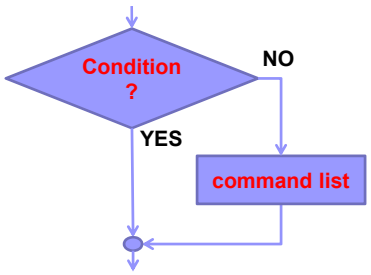
```
original_location=`pwd` ; cd $HOME ; ls ; cd $original_location
```

Use of Colon

- Sometimes it is useful to have a command which does *nothing*
- The : (colon) command in Unix does nothing

```
#!/bin/sh
if grep "UNIX" myfile > /dev/null
then
   :
else
   echo "Sorry, UNIX was not found"
fi
```

The : is needed, when you can not "*complement*" the result of a *decision_command*



The test Command

- test is a Unix command that can
  - Check file type/size
  - Compare strings
  - Compare integer values
- As a result of this check/comparison, it returns
  - Successful
  - Unsuccessful

The test Command - File Tests

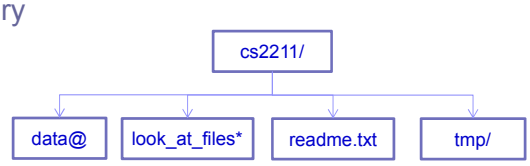
- test -f file ➔ does file exist and is not a directory or a link?
- test -L file ➔ does file exist and is a link?
- test -d file ➔ does file exist and is a directory?
- test -x file ➔ does file exist and the executable flag is on?
- test -s file ➔ does file exist and its size is bigger than 0 bytes?

```
#!/bin/sh
count=0
for i in * ; do
   if test -x "$i" ; then
      count=`expr $count + 1`
   fi
done
echo Total of $count files executable
```

Why did I put double quotation here?

The test Command - File Tests

```
#!/bin/sh
# To look at files in a directory
files=`ls`
for filename in $files
do
   if test -f $filename ; then
      echo "Copying $filename to $filename.bak"
      cp $filename $filename.bak
   fi
done
```




```
obelix[48]% look_at_files
Copying look_at_files to look_at_files.bak
Copying readme.txt to readme.txt.bak
obelix[49]%
```

The test Command - String Tests

- test -z string ➔ does the length of string equal 0?
- test string1 = string2 ➔ does string1 equal string2?
- test string1 != string2 ➔ does string1 not equal string2?
- Example

```
#!/bin/sh
if test -z $REMOTEHOST
then
   :
else
   DISPLAY="$REMOTEHOST:0"
   export DISPLAY
fi
```


The test Command - Integer Tests

- Integers can also be compared
- Use -eq, -ne, -lt, -le, -gt, -ge
- Example

```
#!/bin/sh
smallest=10000
for i in 19 28 5 8 6 3 7 ; do
   if test $i -lt $smallest ; then
      smallest=$i
   fi
done
echo $smallest
```

```
sh -xv find_the_smallest
#!/bin/sh
smallest=10000
smallest=10000
for i in 19 28 5 8 6 3 7 ; do
   if test $i -lt $smallest ; then
      smallest=$i
   fi
done
+ test 19 -lt 10000
smallest=19
+ test 28 -lt 19
+ test 5 -lt 19
smallest=5
+ test 8 -lt 5
+ test 6 -lt 5
+ test 3 -lt 5
smallest=3
+ test 7 -lt 3
echo $smallest
+ echo 3
3
```

Use of [ ]

- The test program has a shorthand as [ ]
- Each bracket must be *surrounded by spaces*
- This is supposed to be a bit easier to read
- For example:

```
#!/bin/sh
smallest=10000
for i in 19 28 5 8 6 3 7 ; do
   if  [  $i -lt $smallest  ]  ; then
      smallest=$i
   fi
done
echo $smallest
```


The while Loop

- while--do--done loops repeat statements ***as long as*** the while condition ***is*** met
- Example

```
#!/bin/sh
i=1
sum=0
while [ $i -le 10  ]
do
   sum=`expr $sum + $i`
   i=`expr $i + 1`
done
echo The sum is $sum
```

Needs end-of-line or a ; separator

Does *not* Need end-of-line separator

Needs end-of-line or a ; separator

Homework : use -xv to trace the above script

The while Loop

- while--do--done loops repeat statements ***as long as*** the while condition ***is*** met
- Example 2:

```
#!/bin/sh
# This script opens 3 terminal windows.

i="0"

while [ $i -lt 3 ]
do
   xterm &
   i=`expr $i + 1`
done
```

The until Loop

- until--do--done loops repeat statements ***till*** the until condition ***is*** met
- Example:

```
#!/bin/sh
# This script opens 3 terminal windows.

i="0"

until [ $i -ge 3 ]
do
   xterm &
   i=`expr $i + 1`
done
```

Needs end-of-line or a ; separator

Does ***not*** Need end-of-line separator

Needs end-of-line or a ; separator

Command Line Arguments

- Shell scripts would not be very useful if we could not pass arguments to them through the command line
- Shell script arguments are *numbered* from left to right
  - $1 - first argument after command
  - $2 - second argument after command
  - $3 - third argument after command
  - ….
  - ….
  - $8 - eighth argument after command
  - $9 - ninth argument after command
  - They are called *positional parameters*
- While more than 9 arguments can be used, only the first 9 can be explicitly referenced by $1, $2, $3, … , $9
- The rest can be accessed by using $* and shift (see next slide)

Command Line Arguments

- Example:
  - Write a command called getline that get a particular line in a file, e.g., getline linenumber filename

```
#!/bin/sh
head -$1 $2 | tail -1
```

- Other variables related to arguments:
  - $0 name of the running command
  - $* All the arguments *starting from* $1 (*even if there are more than* 9)
  - $# the number of arguments, $0 is *not included in the count*
- The shift command
  - shifts all the arguments to the left, i.e., $1 ← $2, $2 ← $3, $3 ← $4, ... ($1 *will be lost*)
  - Decrease the value of $# by one (i.e., $# ← $# - 1)
  - useful when there are more than 9 arguments

Command Line Arguments

- Example:

```
#!/bin/sh
# creates multiple symbolic links, where 1st argument is the target
original=$1
if [ -f $original ]
then
   shift
else
   echo "$original does not exist"
   exit 1
fi
while [ $# -gt 0 ]
do
   ln -s $original $1
   shift
done
```

Homework : use -xv to trace the above script

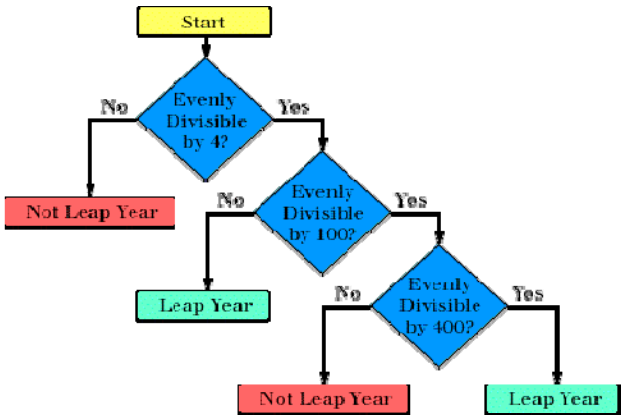
Reading Variables From Standard Input

- The read command reads one line of input *from the standard input* and assigns its words to variables given as arguments
- Syntax: read var1 var2 var3 ...
  - reads a line of input from standard input
  - Assign first word to var1, second word to var2, ...
  - The last variable gets any excess words on the line
  - *What if we have words less than variables?*
- Example:

```
% read X Y Z
Here are some words as input
% echo $X
Here
% echo $Y
are
% echo $Z
some words as input
```

Reading Variables From Standard Input

- Example: To check if you have given a leap year or not.



Reading Variables From Standard Input

- Example: To check if you have given a leap year or not.

```
#!/bin/sh
# This script will test if you have given a leap year or not.
echo "Type the year that you want to check (4 digits), followed by [ENTER]:"
read year
if [ `expr $year % 4` -eq 0 ]; then
    if [ `expr $year % 100` -eq 0 ]; then
        if [ `expr $year % 400` -eq 0 ]; then
            echo "$year is a leap year.  February has 29 days."
        else
            echo "$year is not a leap year.  February has 28 days."
        fi
    else
        echo "$year is a leap year.  February has 29 days."
    fi
else
    echo "$year is not a leap year.  February has 28 days."
fi
```

Reading Variables From Standard Input

- Example: To check if you have given a leap year or not.

```
#!/bin/sh
# This script will test if you have given a leap year or not.
echo "Type the year that you want to check (4 digits), followed by [ENTER]:"
read year
if [ `expr $year % 4` -ne 0 ]; then
    echo "$year is not a leap year.  February has 28 days."
elif [ `expr $year % 100` -eq 0 ]; then
    if [ `expr $year % 400` -eq 0 ]; then
        echo "$year is a leap year.  February has 29 days."
    else
        echo "$year is not a leap year.  February has 28 days."
    fi
else
    echo "$year is a leap year.  February has 29 days."
fi
```


Reading Variables From Standard Input

- Example: To check if you have given a leap year or not.

```
#!/bin/sh
# This script will test if you have given a leap year or not.
echo "Type the year that you want to check (4 digits), followed by [ENTER]:"
read year
if [ `expr $year % 4` -ne 0 ]; then
    echo "$year is not a leap year.  February has 28 days."
else
    if [ `expr $year % 100` -eq 0 ]; then
        if [ `expr $year % 400` -eq 0 ]; then
            echo "$year is a leap year.  February has 29 days."
        else
            echo "$year is not a leap year.  February has 28 days."
        fi
    else
        echo "$year is a leap year.  February has 29 days."
    fi
fi
```

case Statement

- The case--in--esac statement supports multi-way branching, based on the value of a single string
- General form

```
case string in
  pattern1)
    command_set_1
    ;;
  pattern2)
    command_set_2
    ;;
  …
esac
```

Does *not* Need end-of-line separator

Does *not* Need end-of-line separator

Does *not* Need end-of-line separator

Does *not* Need end-of-line separator

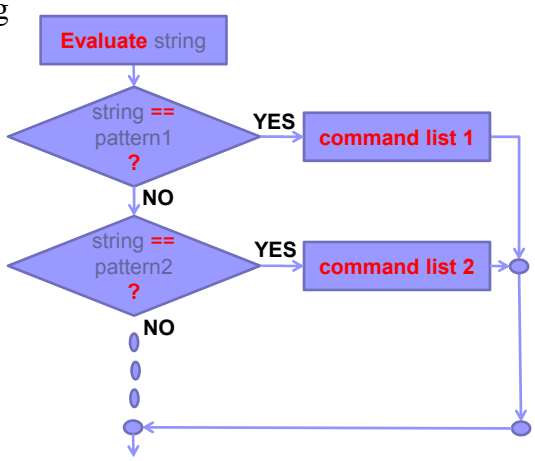
No space between ;;

- Wild carding (i.e., *, ?, [ ]) can be used in patterns

case Statement

- The case--in--esac statement supports multi-way branching, based on the value of a single string
- General form

```
case string in
  pattern1)
    command_set_1
    ;;
  pattern2)
    command_set_2
    ;;
  …
esac
```

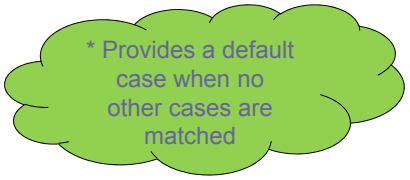


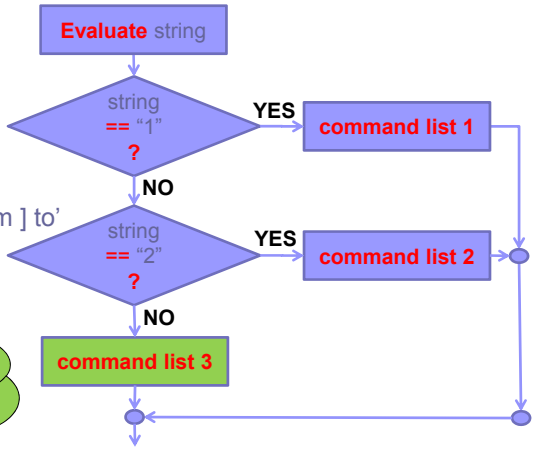
- Wild carding (i.e., *, ?, [ ]) can be used in patterns

case Statement

- Example:

```
#!/bin/sh
case $# in
    1)
        cat >>$1
        ;;
    2)
        cat >>$2 <$1
        ;;
    *)
        echo 'usage: append [ from ] to'
        ;;
esac
```





case Statement

- This is an append command
- When called with one argument as
  append my-file
  - $# will be the string 1
  - the *standard input* is copied onto the end of my-file using the cat command
- When called with two argument as
  append my-file-1 my-file-2
  - $# will be the string 2
  - my-file-1 is copied onto the end of my-file-2 using the cat command
- If the number of arguments supplied to the append command is anything other than 1 or 2
  - a message is printed indicating proper usage

case Statement

- Case statement can by written as follow:

```
case $# in
    1)
       cat >>$1
       ;;
    2)
       cat >>$2 <$1
       ;;
    *)
       echo 'usage: append [ from ] to'
       ;;
 esac
```

Or

```
case $# in
    1) cat >>$1;;
    2) cat >>$2 <$1;;
    *) echo 'usage: append [ from ] to' ;;
 esac
```

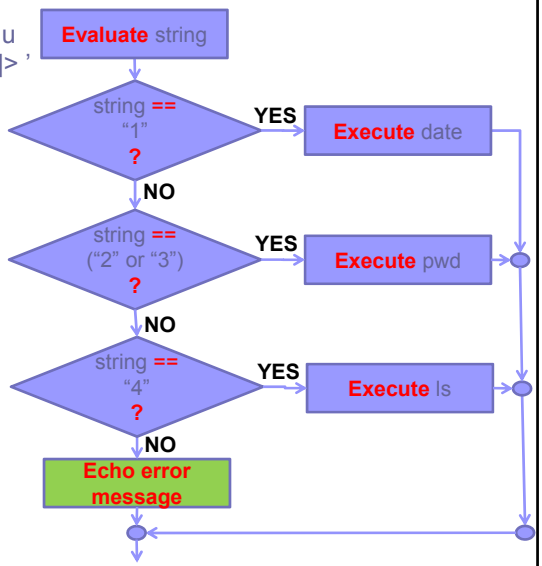
Or

```
case $# in  1) cat >>$1;;  2) cat >>$2 <$1;;  *) echo 'usage: append [ from ] to' ;; esac
```


case Statement

```
#!/bin/sh
# Uses case to offer 4-item menu
echo -n 'Choose command [1-4]> '
read reply
echo
case $reply in
       1)
          date
          ;;
       [23])
          pwd
          ;;
       4)
          ls
          ;;
       *)
          echo Illegal choice!
          ;;
esac
```

What will happen if we do not have ;; here?



case Statement

- ***Caution***: no check is made to ensure that only one pattern matches the case argument
- If more than one match occur, only the first one will be executed
- In the example below the commands following the second * will never be executed
- case $# in

```
        .....

        *) ... ;;
        *) ... ;;
    esac
```


case Statement

- To allow the same commands to be associated with more than one pattern, the case command provides alternative patterns separated by a |
- For example,

```
    case $i in
        -x|-y)    ... ;;
    esac
```

is equivalent to

```
    case $i in
        -[xy])    ... ;;
    esac
```

case Statement

- The usual quoting conventions apply
- Example

```
case $i in
    ?)      ... ;;
```

will match any single character

```
case $i in
        \?)     ... ;;
```

will match the character ?

A Shell Script Example

- Suppose we have a file called marks.txt containing the following student grades:

```
091286899  90  H. White
197920499  80  J. Brown
899268899  75  A. Green
……
```

- We want to calculate some statistics on the grades in this file

A Shell Script Example

```
#!/bin/sh
sum=0; count=0; count_fail=0
while read  student_num  grade  name
do
   sum=`expr $sum + $grade`
   count=`expr $count + 1`
   if [ $grade -lt 50 ]
   then
     countfail=`expr $countfail + 1`
   fi
done
echo The average is `expr $sum / $count`
echo $countfail students failed
```

- *Homework* : use -xv to trace the above script

A Shell Script Example

- Suppose the previous shell script was saved in a file called statistics and its permission was set to be execute
- To run this script, you can
  - cat marks.txt | statistics
  - statistics < marks.txt
  - execute statistics and provide marks through standard input, ending by Ctrl-d