Decision Making & Loop Control

Example

```
#!/bin/bash
  if [ -f /etc/fstab ];
  then
            cp /etc/fstab .
            echo “Done.”
  else
            echo “This file does not exist.”
            exit 1
  fi
```

Exercise.

- Write a shell script which:
  - accepts a file name
  - checks if file exists
  - if file exists, copy the file to the same name + the current date (if the backup file already exists ask if you want to replace it).
- When done you should have the original file and one with a current date at the end.

Expressions

- Logical operators:

!	negate (NOT) a logical expression
-a	logically AND two logical expressions
-o	logically OR two logical expressions

Example:

```
#!/bin/bash
  echo -n “Enter a number 1 < x < 10:”
  read num
  if [ “$num” -gt 1 -a “$num” -lt 10 ];
  then
            echo “$num*$num=$(($num*$num))”
  else
            echo “Wrong insertion !”
  fi
```

Expressions

- Logical operators:

&&	logically AND two logical expressions
||	logically OR two logical expressions

Example:

```
#!/bin/bash
  echo -n "Enter a number 1 < x < 10: "
  read num
  if [ “$number” -gt 1 ] && [ “$number” -lt 10 ];
  then
            echo “$num*$num=$(($num*$num))”
  else
            echo “Wrong insertion !”
  fi
```

Example

$ cat iftrue.sh

```
#!/bin/bash
echo “Enter a path: ”; read x
if cd $x; then
        echo “I am in $x and it contains”; ls
else
        echo “The directory $x does not exist”;
        exit 1
fi
```

Shell Parameters

- Positional parameters are assigned from the shell's argument when it is invoked. Positional parameter "N" may be referenced as "${N}", or as "$N" when "N" consists of a single digit.

- Special parameters

$#	is the number of parameters passed
$0	returns the name of the shell script running as well as its location in the file system
$*	gives a single word containing all the parameters passed to the script
$@	gives an array of words containing all the parameters passed to the script

```
$ cat sparameters.sh
#!/bin/bash
echo "$#; $0; $1; $2; $*; $@"
$ sparameters.sh arg1 arg2
2; ./sparameters.sh; arg1; arg2; arg1 arg2; arg1 arg2
```

Trash

```
$ cat trash.sh
  #!/bin/bash
  if [ $# -eq 1 ];
  then
            if [ ! –d “$HOME/trash” ];
            then
                        mkdir “$HOME/trash”
            fi
            mv $1 “$HOME/trash”
  else
            echo “Use: $0 filename”
            exit 1
  fi
```

Case Statement

- Used to execute statements based on specific values. Often used in place of an if statement if there are a large number of conditions.

- Value used can be an expression
- each set of statements must be ended by a pair of semicolons;
- a *) is used to accept any value not matched with list of values

```
case $var in
  val1)
        statements;;
  val2)
        statements;;
  *)
        statements;;
  esac
```

Example 1 (case.sh)

```
$ cat case.sh
#!/bin/bash
  echo -n “Enter a number 1 < x < 10: ”
  read x
  case $x in
            1) echo “Value of x is 1.”;;
            2) echo “Value of x is 2.”;;
            3) echo “Value of x is 3.”;;
            4) echo “Value of x is 4.”;;
            5) echo “Value of x is 5.”;;
            6) echo “Value of x is 6.”;;
            7) echo “Value of x is 7.”;;
            8) echo “Value of x is 8.”;;
            9) echo “Value of x is 9.”;;
            0 | 10) echo “wrong number.”;;
            *) echo “Unrecognized value.”;;
  esac
```

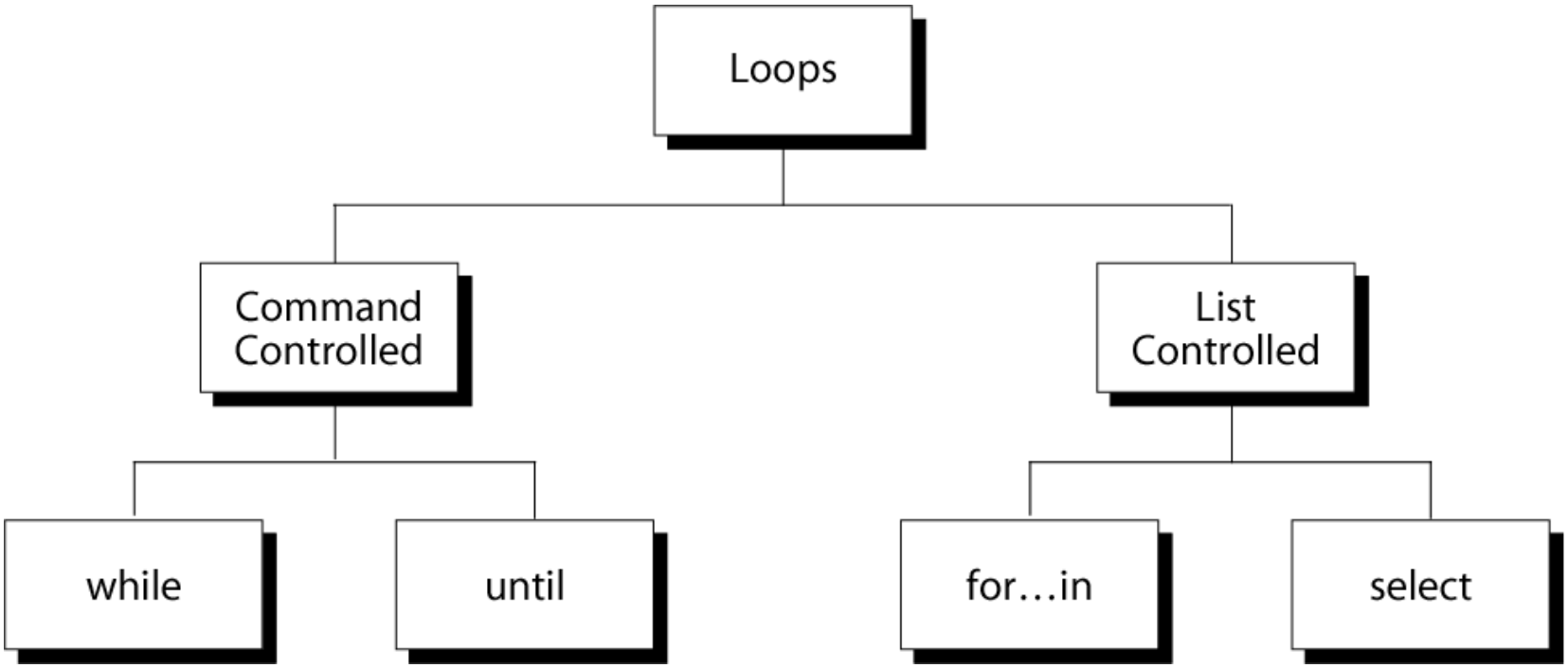
Example 2: The case Statement

```
#!/bin/bash
echo "Enter Y to see all files including hidden files"
echo "Enter N to see all non-hidden files"
echo "Enter q to quit"

read -p "Enter your choice: " reply

case $reply in
  Y|YES) echo "Displaying all (really…) files"
         ls -a ;;
  N|NO)  echo "Display all non-hidden files..."
         ls ;;
  Q)     exit 0 ;;

  *) echo "Invalid choice!"; exit 1 ;;
esac
```

Example 3: The case Statement

```
#!/bin/bash
ChildRate=3
AdultRate=10
SeniorRate=7
read -p "Enter your age: " age
case $age in
  [1-9]|[1][0-2])   # child, if age 12 and younger
     echo "your rate is" '$'"$ChildRate.00" ;;
 # adult, if age is between 13 and 59 inclusive
  [1][3-9]|[2-5][0-9])
     echo "your rate is" '$'"$AdultRate.00" ;;
  [6-9][0-9])       # senior, if age is 60+
     echo "your rate is" '$'"$SeniorRate.00" ;;
esac
```

Repetition Constructs



Iteration Statements

- The for structure is used when you are looping through a range of variables.

```
for var in list
  do
    statements
  done
```

- statements are executed with var set to each value in the list.

- Example

```
#!/bin/bash
  let sum=0
  for num in 1 2 3 4 5
     do
       let “sum = $sum + $num”
     done
  echo $sum
```

Iteration Statements

```
#!/bin/bash
  for x in paper pencil pen
  do
    echo “The value of variable x is: $x”
    sleep 1
  done
```

- if the list part is left off, var is set to each parameter passed to the script ($1, $2, $3,…)

```
$ cat for1.sh
  #!/bin/bash
  for x
  do
    echo “The value of variable x is: $x”
    sleep 1
  done
$ for1.sh arg1 arg2
  The value of variable x is: arg1
  The value of variable x is: arg2
```

Example (old.sh)

```
$ cat old.sh
#!/bin/bash
# Move the command line arg files to old directory.
if [ $# -eq 0 ] #check for command line arguments
then
  echo “Usage: $0 file …”
  exit 1
fi
if [ ! –d “$HOME/old” ]
then
  mkdir “$HOME/old”
fi
echo The following files will be saved in the old directory:
echo $*
for file in $* #loop through all command line arguments
do
  mv $file “$HOME/old/”
  chmod 400 “$HOME/old/$file”
done
ls -l “$HOME/old”
```

Example (args.sh)

```
$ cat args.sh
#!/bin/bash
# Invoke this script with several arguments: “one two three“
if [ ! -n “$1” ]; then
  echo “Usage: $0 arg1 arg2 ..." ; exit 1
fi
echo ; index=1 ;
echo “Listing args with \”\$*\”:”
for arg in “$*” ;
do
  echo “Arg $index = $arg”
  let “index+=1” # increase variable index by one
done
echo “Entire arg list seen as single word.”
echo ; index=1 ;
echo “Listing args with \”\$@\”:”
for arg in “$@” ; do
  echo “Arg $index = $arg”
  let “index+=1”
done
echo “Arg list seen as separate words.” ; exit 0
```

Logic: for

```
for i in /*
do
  echo “Listing $i:”
  ls -l $i
  read
done
```

```
#!/bin/bash

for i in 7 9 2 3 4 5
do
   echo $i
done
```

Example 2: Using the for Loop

```
#!/bin/bash
# compute the average weekly temperature

for num in 1 2 3 4 5 6 7
do
   read -p "Enter temp for day $num: " Temp
   let TempTotal=$TempTotal+$Temp
done

let AvgTemp=$TempTotal/7
echo "Average temperature: " $AvgTemp
```

A C-like for loop

- An alternative form of the for structure is

```
for (( EXPR1 ; EXPR2 ; EXPR3 ))
  do
            statements
  done
```

- First, the arithmetic expression EXPR1 is evaluated. EXPR2 is then evaluated repeatedly until it evaluates to 0. Each time EXPR2 is evaluates to a non-zero value, statements are executed and EXPR3 is evaluated.

```
$ cat for2.sh
  #!/bin/bash
  echo –n “Enter a number: ”; read x
  let sum=0
  for (( i=1 ; $i<$x ; i=$i+1 )) ; do
    let “sum = $sum + $i”
  done
  echo “the sum of the first $x numbers is: $sum”
```

While Statements

- The while structure is a looping structure. Used to execute a set of commands while a specified condition is true. The loop terminates as soon as the condition becomes false. If condition never becomes false, loop will never exit.

```
while expression
  do
            statements
  done
```

```
$ cat while.sh
  #!/bin/bash
  echo –n “Enter a number: ”; read x
  let sum=0; let i=1
  while [ $i –le $x ]; do
    let “sum = $sum + $i”
              i=$i+1
  done
  echo “the sum of the first $x numbers is: $sum”
```

Example: Using the while Loop

```
#!/bin/bash
COUNTER=0
while [ $COUNTER -lt 10 ]
do
    echo The counter is $COUNTER
    let COUNTER=$COUNTER+1
done
```

Menu

```
$ cat menu.sh
#!/bin/bash
  clear ; loop=y
  while [ “$loop” = y ] ;
  do
    echo “Menu”;  echo “====”
                  echo “D: print the date”
    echo “W: print the users who are currently log on.”
    echo “P: print the working directory”
    echo “Q: quit.”
    echo
    read –s choice                  # silent mode: no echo to terminal
    case $choice in
                D | d) date ;;
                W | w) who ;;
                P | p) pwd ;;
                Q | q) loop=n ;;
                *) echo “Illegal choice.” ;;
    esac
    echo
  done
```

Find a Pattern and Edit

```
$ cat grepedit.sh
#!/bin/bash
# Edit argument files $2 ..., that contain pattern $1
if [ $# -le 1 ]
then
  echo “Usage: $0 pattern file …” ; exit 1
else
  pattern=$1                    # Save original $1
  shift                         # shift the positional parameter to the left by 1
  while [ $# -gt 0 ]            # New $1 is first filename
  do
    grep “$pattern” $1 > /dev/null
    if [ $? -eq 0 ] ; then      # If grep found pattern
      vi $1                     # then vi the file
    fi
    shift
  done
fi
$ grepedit.sh while ~
```

Example: Using the while Loop

```
#!/bin/bash
# copies files from home- into the webserver-
  directory
# A new directory is created every hour

PICSDIR=/home/carol/pics
WEBDIR=/var/www/carol/webcam
while true; do
   DATE=`date +%Y%m%d`
   HOUR=`date +%H`
   mkdir $WEBDIR/"$DATE"
   while [ $HOUR -ne "00" ]; do
      DESTDIR=$WEBDIR/"$DATE"/"$HOUR"
      mkdir "$DESTDIR"
      mv $PICSDIR/*.jpg "$DESTDIR"/
      sleep 3600
      HOUR=`date +%H`
   done
done
```

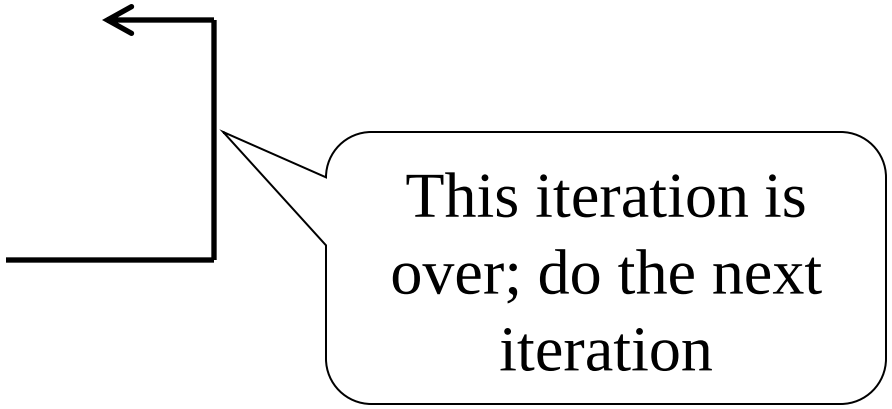
Continue Statements

- The continue command causes a jump to the next iteration of the loop, skipping all the remaining commands in that particular loop cycle.

```
$ cat continue.sh
#!/bin/bash
  LIMIT=19
  echo
  echo “Printing Numbers 1 through 20 (but not 3 and 11)”
  a=0
  while [ $a -le “$LIMIT” ]; do
    a=$(($a+1))
    if [ “$a” -eq 3 ] || [ “$a” -eq 11 ]
    then
              continue
    fi
    echo -n “$a ”
  done
```

The continue

```
while [ condition ]
do
     cmd-1
     continue
     cmd-n
done
echo "done"
```



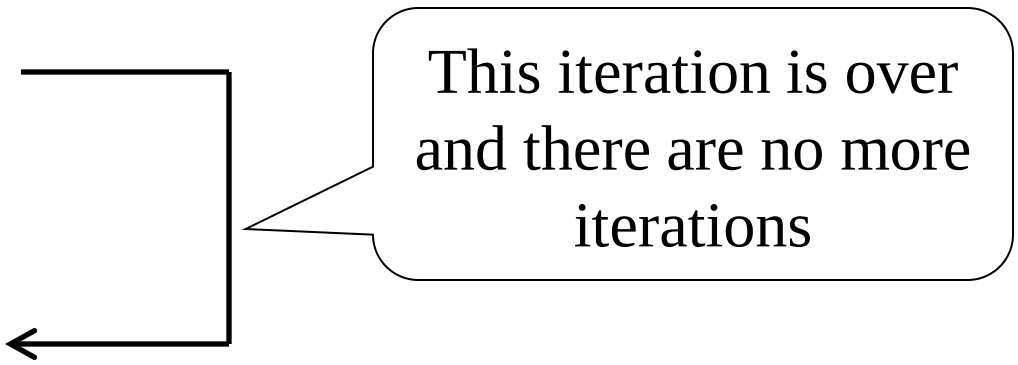
Break Statements

- The break command terminates the loop (breaks out of it).

```
$ cat break.sh
#!/bin/bash
  LIMIT=19
  echo
  echo “Printing Numbers 1 through 20, but something happens after 2 … ”
  a=0
  while [ $a -le “$LIMIT” ]
  do
    a=$(($a+1))
    if [ “$a” -gt 2 ]
    then
      break
    fi
    echo -n “$a ”
  done
  echo; echo; echo
  exit 0
```

The break

```
while [ condition ]
do
     cmd-1
     break
     cmd-n
done
echo "done"
```



Example:

```
for index in 1 2 3 4 5 6 7 8 9 10
do
        if [ $index -le 3 ]; then
              echo "continue"
              continue
        fi
        echo $index
        if   [ $index -ge 8 ]; then
              echo "break"
              break
        fi
done
```

Until Statements

- The until structure is very similar to the while structure. The until structure loops until the condition is true. So basically it is “until this condition is true, do this”.

```
until [expression]
  do
            statements
  done
```

```
$ cat countdown.sh
  #!/bin/bash
  echo “Enter a number: ”; read x
  echo ; echo Count Down
  until [ “$x” -le 0 ]; do
    echo $x
    x=$(($x –1))
    sleep 1
  done
echo ; echo GO !
```

Example: Using the until Loop

```
#!/bin/bash

Stop="N"
until [ $Stop = "Y" ]; do
  ps -A
  read -p "want to stop? (Y/N)" reply
  Stop=`echo $reply | tr [:lower:] [:upper:]`
done
echo "done"
```

Manipulating Strings

- Bash supports a number of string manipulation operations.

${#string} gives the string length
${string:position} extracts sub-string from $string at $position
${string:position:length} extracts $length characters of sub-string from $string at $position

- Example

```
$ st=0123456789
$ echo ${#st}
  10
$ echo ${st:6}
  6789
$ echo ${st:6:2}
  67
```

Parameter Substitution

- Manipulating and/or expanding variables

${parameter-default}, if parameter not set, use default.

```
$ echo ${username-`whoami`}
  alice
$ username=bob
$ echo ${username-`whoami`}
  bob
```

${parameter=default}, if parameter not set, set it to default.

```
$ unset username
$ echo ${username=`whoami`}
$ echo $username
  alice
```

${parameter+value}, if parameter set, use value, else use null string.

```
$ echo ${username+bob}
  bob
```

Parameter Substitution

${parameter?msg}, if parameter set, use it, else print msg

```
$ value=${total?'total is not set'}
  total: total is not set
$ total=10
$ value=${total?'total is not set'}
$ echo $value
  10
```

Example

```
#!/bin/bash
OUTFILE=symlinks.list                    # save file
directory=${1-`pwd`}
for file in "$( find $directory -type l )"
                                         # -type l == symbolic links
do
  echo "$file"
done | sort >> "$HOME/$OUTFILE"
exit 0
```