```
#!/bin/bash
# planets.sh : for loop with two parameters in each [list] element
# Associate the name of each planet with its distance from the sun.
for planet in "Mercury 36" "Venus 67" "Earth 93" "Mars 142" "Jupiter 483"
do
        set -- $planet  # Parses variable "planet" and sets positional parameters.
                        # The "--" prevents nasty surprises if $planet is null or begins with a dash.
                        # May need to save original positional parameters, since they get overwritten.
                        # One way of doing this is to use an array,
                        #
        original_params=("$@")
        echo "$1        $2,000,000 miles from the sun"
        #-------two     tabs---concatenate zeroes onto parameter $2
done
exit 0
```

```
#!/bin/bash
# recurse.sh : A (useless) script that recursively calls itself
# Can a script recursively call itself?
# Yes, but is this of any practical use?
# (See the following.)
RANGE=10
MAXVAL=9
i=$RANDOM
let "i %= $RANGE"   # Generate a random number between 0 and $RANGE - 1.
if [ "$i" -lt "$MAXVAL" ]
then
        echo "i = $i"
        ./$0              # Script recursively spawns a new instance of itself.
fi
# Each child script does the same, until a generated $i equals $MAXVAL.
exit 0

# Note:
# ------
# This script must have execute permission for it to work properly.
# This is the case even if it is invoked by an "sh" command.
```

```
#!/bin/bash
# pb.sh: phone book : A (useful) script that recursively calls itself
# Written by Rick Boivie, and used with permission.
# Modifications by ABS Guide author.
MINARGS=1           # Script needs at least one argument.
DATAFILE=./phonebook      # A data file in current working directory named "phonebook" must exist.
PROGNAME=$0
E_NOARGS=70         # No arguments error.
```

```
if [ $# -lt $MINARGS ]; then
        echo "Usage: "$PROGNAME" data-to-look-up"
        exit $E_NOARGS
fi
if [ $# -eq $MINARGS ]; then
        grep $1 "$DATAFILE" # 'grep' prints an error message if $DATAFILE not present.
else
        ( shift; "$PROGNAME" $* ) | grep $1        # Script recursively calls itself.
fi
exit 0          # Script exits here.
                # Therefore, it's o.k. to put non-hashmarked comments and data after this point.
# ---------------------------------------------------------------------
Sample "phonebook" datafile:
John Doe      1555 Main St., Baltimore, MD 21228      (410) 222-3333
Mary Moe      9899 Jones Blvd., Warren, NH 03787      (603) 898-3232
Richard Roe   856 E. 7th St., New York, NY 10009      (212) 333-4567
Sam Roe       956 E. 8th St., New York, NY 10009      (212) 444-5678
Zoe Zenobia   4481 N. Baker St., San Francisco, SF     94338 (415) 501-1631
```

```
$ cat friends.sh
#!/bin/bash
# This is a program that keeps your address book up to date.
friends="/var/tmp/michel/friends"
echo "Hello, "$USER". This script will register you in Michel's friends database."
echo -n "Enter your name and press [ENTER]: "
read name
echo -n "Enter your gender and press [ENTER]: "
read -n 1 gender
grep -i "$name" "$friends"
if [ $? == 0 ]; then
        echo "You are already registered, quitting."
        exit 1
elif [ "$gender" == "m" ]; then
        echo "You are added to Michel's friends list."
        exit 1
else
        echo -n "How old are you? "
        read age
        if [ $age -lt 25 ]; then
                echo -n "Which colour of hair do you have? "
                read colour
                echo "$name $age $colour" >> "$friends"
                echo "You are added to Michel's friends list. Thank you so much!"
        else
                echo "You are added to Michel's friends list."
                exit 1
        fi
fi
-------------------------------------------------------
```

$ cp friends.sh /var/tmp; cd /var/tmp
$ touch friends; chmod a+w friends
$ friends.sh
Hello, michel. This script will register you in Michel's friends database.
Enter your name and press [ENTER]: michel
Enter your gender and press [ENTER] :m
You are added to Michel's friends list.
$ cat friends

```
#!/bin/bash
# ascript : variables inside a function
function afunc
{
        echo in function: $0 $1 $2
        var1="in function"
        echo var1: $var1
}
var1="outside function"
echo var1: $var1
echo $0: $1 $2
afunc funcarg1 funcarg2
echo var1: $var1
echo $0: $1 $2
----------------------------
```

$ ascript arg1 arg2

```
# Script to create simple menus and take action according to that selected
# menu item
#
while :
 do
    clear
    echo "-------------------------------------"
    echo " Main Menu "
    echo "-------------------------------------"
    echo "[1] Show Todays date/time"
    echo "[2] Show files in current directory"
    echo "[3] Show calendar"
    echo "[4] Start editor to write letters"
    echo "[5] Exit/Stop"
    echo "======================"
    echo -n "Enter your menu choice [1-5]: "
    read yourch
    case $yourch in
      1) echo "Today is `date` , press a key. . ." ; read ;;
      2) echo "Files in `pwd`" ; ls -l ; echo "Press a key. . ." ; read ;;
      3) cal ; echo "Press a key. . ." ; read ;;
      4) vi ;;
      5) exit 0 ;;
      *) echo "Opps!!! Please select choice 1,2,3,4, or 5";
         echo "Press a key. . ." ; read ;;
 esac
done
```

```
#!/bin/bash
# ascii.sh        : This script generates ASCII table
# ver. 0.2, reldate 26 Aug 2008
# Patched by ABS Guide author.
# Original script by Sebastian Arming.
#
exec >ASCII.txt         # Save stdout to file.
MAXNUM=256
COLUMNS=5
OCT=8
OCTSQU=64
LITTLESPACE=-3
BIGSPACE=-5
i=1      # Decimal counter
o=1      # Octal counter
while [ "$i" -lt "$MAXNUM" ]; do   # We don't have to count past 400 octal.
        paddi="          $i"
        echo -n "${paddi: $BIGSPACE} "    # Column spacing.
        paddo="00$o"
        echo -ne "\\0${paddo: $LITTLESPACE}"
        echo -n  "       "
        if (( i % $COLUMNS == 0)); then
                # New line.
                echo
        fi
        ((i++, o++))
        # The octal notation for 8 is 10, and 64 decimal is 100 octal.
        (( i % $OCT == 0))    && ((o+=2))
        (( i % $OCTSQU == 0)) && ((o+=20))
done
exit $?
```

```
#!/bin/bash
# days-between.sh: Number of days between two dates.
# Usage: ./days-between.sh [M]M/[D]D/YYYY [M]M/[D]D/YYYY
#
ARGS=2                  # Two command-line parameters expected.
E_PARAM_ERR=85          # Param error.
REFYR=1600              # Reference year.
CENTURY=100
DIY=365
ADJ_DIY=367             # Adjusted for leap year + fraction.
MIY=12
DIM=31
LEAPCYCLE=4
MAXRETVAL=255           # Largest permissible positive return value from a function.
diff=                   # Declare global variable for date difference.
value=                  # Declare global variable for absolute value.
day=                    # Declare globals for day, month, year.
```

```
month=
year=
#
Param_Error ()                  # Command-line parameters wrong.
{
        echo "Usage: `basename $0` [M]M/[D]D/YYYY [M]M/[D]D/YYYY"
        echo "           (date must be after 1/3/1600)"
        exit $E_PARAM_ERR
}
#
Parse_Date ()                   # Parse date from command-line params.
{
        month=${1%%/**}
        dm=${1%/**}             # Day and month.
        day=${dm#*/}
        let "year = `basename $1`"    # Not a filename, but works just the same.
}
#
check_date ()                   # Checks for invalid date(s) passed.
{
        [ "$day" -gt "$DIM" ] || [ "$month" -gt "$MIY" ] ||
        [ "$year" -lt "$REFYR" ] && Param_Error
        # Exit script on bad value(s).
        # Uses or-list / and-list.
}
#
strip_leading_zero ()           # Better to strip possible leading zero(s) from day and/or month
{
        return ${1#0}           # since otherwise Bash will interpret them as octal values
}
#
day_index ()            # Gauss' Formula: # Days from March 1, 1600 to date passed as param.
{
        day=$1
        month=$2
        year=$3
        let "month = $month - 2"
        if [ "$month" -le 0 ]
        then
                let "month += 12"
                let "year -= 1"
        fi
        let "year -= $REFYR"
        let "indexyr = $year / $CENTURY"
        let "Days = $DIY*$year + $year/$LEAPCYCLE - $indexyr  \
                + $indexyr/$LEAPCYCLE + $ADJ_DIY*$month/$MIY + $day - $DIM"
        echo $Days
}
#
```

```
calculate_difference ()                 # Difference between two day indices.
{
        let "diff = $1 - $2"            # Global variable.
}
#
abs ()                                  # Absolute value
{                                       # Uses global "value" variable.
        if [ "$1" -lt 0 ]               # If negative
        then
                let "value = 0 - $1"    # change sign,
        else
                let "value = $1"
        fi
}
#
#
#
if [ $# -ne "$ARGS" ]                   # Require two command-line params.
then
        Param_Error
fi
Parse_Date $1
check_date $day $month $year            # See if valid date.
strip_leading_zero $day                 # Remove any leading zeroes on day and/or month.
day=$?
strip_leading_zero $month
month=$?
#
let "date1 = `day_index $day $month $year`"
Parse_Date $2
check_date $day $month $year
strip_leading_zero $day
day=$?
strip_leading_zero $month
month=$?
#
date2=$(day_index $day $month $year)      # Command substitution.
calculate_difference $date1 $date2

abs $diff                               # Make sure it's positive.
diff=$value
echo $diff
exit 0
```

```
#!/bin/bash
# passgen.sh : Random password generator
MATRIX="0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"
                # Password will consist of alphanumeric characters.
LENGTH="8"          # ==> May change 'LENGTH' for longer password.
```

```
while [ "${n:=1}" -le "$LENGTH" ]          # ==> Recall that := is "default substitution" operator.
                                           # ==> So, if 'n' has not been initialized, set it to 1.
do
        PASS="$PASS${MATRIX:$(($RANDOM%${#MATRIX})):1}"

        # Very clever, but tricky.
        # Starting from the innermost nesting...
        # ${#MATRIX} returns length of array MATRIX.
        # $RANDOM%${#MATRIX} returns random number between 1 and [length of MATRIX] - 1.

        # ${MATRIX:$(($RANDOM%${#MATRIX})):1} returns expansion of MATRIX at random
        # position, by length 1.

        # PASS=... simply pastes this result onto previous PASS (concatenation).
        # To visualize this more clearly, uncomment the following line
        #       echo "$PASS"
        # to see PASS being built up, one character at a time, each iteration of the loop.

        let n+=1        # Increment 'n' for next pass.
done
echo "$PASS"            # Or, redirect to a file, as desired.
exit 0
```

```
#!/bin/bash
# primes.sh: Generate prime numbers, without using arrays.
# Script contributed by Stephane Chazelas.
# This does *not* use the classic "Sieve of Eratosthenes" algorithm,
#+ but instead the more intuitive method of testing each candidate number
#+ for factors (divisors), using the "%" modulo operator.
#
LIMIT=1000                      # Primes, 2 ... 1000.
Primes()
{
        (( n = $1 + 1 ))               # Bump to next integer.
        shift
        # echo "_n=$n i=$i_"         # Next parameter in list.
        if (( n == LIMIT ))
        then
                echo $*
                return
        fi
#
        for i; do                      # "i" set to "@", previous values of $n.
                # echo "-n=$n i=$i-"
                (( i * i > n )) && break       # Optimization.
                (( n % i )) && continue        # Sift out non-primes using modulo operator.
                Primes $n $@                   # Recursion inside loop.
                return
        done
```

```
        Primes $n $@ $n                 # Recursion outside loop.
                                        # Successively accumulate
                                        # positional parameters.
                                        # "$@" is the accumulating list of primes.
}

Primes 1

exit $?

# Pipe output of the script to 'fmt' for prettier printing.
# Uncomment lines 16 and 24 to help figure out what is going on.
```