Arithmetic

Use the **expr** command to perform calculations:

e.g.
```
$ x=8
$ y=5
$ expr $x + $y                    (note that spaces required either side of + sign)
13
```

Take care with * as this has another meaning in the shell – i.e.
```
$ expr $x "*" $y
```

Read/Arithmetic Example Script

```
$ nano addition
```

```
# Take two numbers from the user and print their sum
echo Enter the first number
read num1
echo Enter the second number
read num2
echo
echo The sum is:
expr $num1 + $num2
```

```
$ chmod +x addition
$ ./addition
```

Decision Control

Use the **if/then/else/fi** commands.

The general format is:

```
if [ condition ]                  (note that spaces required either side of square brackets)
then
      commands
else                              (else part is optional)
      commands
fi
```

Decision control Example Script

```
$ nano rmfiles
```

```
# This script removes all files in the ~/temp directory
echo "This will remove all files in the temp directory!"
echo "Are you sure (y/n)?"
read response
if [ $response = "y" ]
then
  rm ~/temp/*
  echo Files removed
```

```
else
  echo Not removed
fi
```

```
$ chmod +x rmfiles
$ ./rmfiles
```

More on Conditions

For comparing strings, use

=	equals	(surrounded by spaces)
!=	not equals	

For comparing integers, use:

-eq	equal to
-ne	not equal to
-lt	less than
-le	less than or equal to
-gt	greater than
-ge	greater than or equal to

Example: This script tells if there are more than 50 users logged on

```
numusers=`who|wc –l`
if [ $numusers –gt 50 ]
then
      echo "More than 50 users!"
fi
```

Note: The case instruction provides an alternative kind of decision control.

Loops:

Programming languages have loop constructs to allow a block of code to be executed repeatedly, either a fixed number of times, or while some condition holds.

`while` instruction general format

```
while [ condition ]
do
      commands
done
```

Example: The following script prints all numbers from 1 to 100

```
# Prints all numbers from 1 to 100
#
num=1
while [ $num -le 100 ]
do
  echo $num
  num=`expr $num + 1`
done
```

An alternative is to use the **`until`** statement
e.g replace the 4th line above with **`until [ $num -gt 100 ]`**

Another kind of loop is the **`for`** loop

General format:

```
for var in varlist
do
      commands
done
```

The **`for`** statement executes the commands once for each value in the list

for example 1

```
# prints the numbers from 101 to 105
#
for num in 1 2 3 4 5
do
    expr $num + 100
done
```

for example 2

```
# removes all Java source files in current directory. Use with care!
#
for filename in *.java
do
    rm $filename
done
```

Other Shell Programming Examples

For more examples, you might like to look at either of the following URLs:

- http://www.shelldorado.com/
- http://www.tldp.org/LDP/Bash-Beginners-Guide/html/

Other shell languages

The default shell on Linux is called **bash** (stands for "Bourne-again shell" as it was designed to replace the "Bourne shell" in Unix).

You can verify that you are running the bash shell by typing echo $SHELL.

Note that some systems may run a different shell by default. Different shells have different shell programming syntax and a script written for bash might not necessarily work on a different shell (e.g. the original Bourne shell - you can run that shell by typing **sh**).