The **while** loop

```
while commands1; do commands2; done
```

Executes `commands2` as long as the last command in `commands1` is successful (i.e. its exit code is 0).
i=1
while [ $i -le 10 ]
do
    echo "$i"
i=$(($i+1))
done

This loop prints all numbers 1 to 10.
Until loop

until commands1 ; do commands2 ; done

Executes commands2 as long as commands1 is unsuccessful (i.e. its exit code is not 0).
i=1
until [ $i -ge 11 ]
do
    echo i is $i
    i=$((i+1))
done
Reading in input from the user

You can ask the user for input by using the `read` command

```
read
read varname

- Asks the user for input
- By default stores the input in $REPLY
- Can read in multiple variables `read x y z`
- `-p` option allows you to print some text
```

Example:
```
read -p "How many apples do you have? " apples
How many apples do you have? 5
$ echo $apples
5
```
read can be used to go line by line through a file:

**Examples:**

```bash
cat f.txt | while read LINE ; do echo $LINE ; done
```
- Prints the contents of `f.txt` line by line (read via pipe).

```bash
while read LINE ; do echo $LINE ; done < f.txt
```
- Prints the contents of `f.txt` line by line (read via redirection)
read can be used to go line by line through any other kind of input:

Examples:

```bash
ls *.txt | while read LINE ; do name=$(echo $LINE | \ sed 's/txt/text/'); mv -v "$LINE" "$(name)" ; done
```

- Renames all .txt files in the current directory as .text files.
The almighty for loop

for loop

for var in list ; do
commands
done
for i in 1 2 3 4; do echo $i; done

for i in {1..4}; do echo $i; done

for i in *; do echo $i; done
#! /bin/bash
# lcountgood.sh
# counts number of lines in a collection of files
i=0
for f in "$@
    do
        j='wc -l < $f'
        i=$(($i+$j))
    done

Recall that $@ expands to all arguments individually quoted ("arg1" "arg2" etc).
What happens if we change $@ to $*? Recall that $* expands to all arguments quoted together ("arg1 arg2 arg3")

```bash
#!/bin/bash
# lcountbad.sh
i="0"
for f in "$*"
do
    j='wc -l < $f'
    i=$((i + j))
done
echo $i
```

This does not work! Let's look at why.
Why we don’t like $*

```bash
#!/bin/bash
# explaingood.sh

count=0
for i in "$@"; do
    let count++
    echo $i
done
done
echo $count
```

This simply echos all the files you pass to the script and how many.

```
$ ./explaingood.sh *
explainbad.sh
explaingood.sh
lcountright.sh
3
```
Why we don’t like $*

But if we change to $*

#! /bin/bash
# explainbad.sh
count=0
for i in "$*" ; do
  let count++
  echo $i
done
echo $count

This simply echos all the files at once and the number 1:

$ ./explaingood.sh *
explainbad.sh explaingood.sh lcoutntright.sh
1
We can also do things like:

```bash
for i in $(seq 1 2 20)
do
    echo $i
done
```

1
3
5
7
9
11
13
15
17
19
even more for loop syntax!

C style:

```bash
for (( c=1; c<=5; c++))
do
    echo $c
done

warning: only in recent bash versions
```
We can now create infinite for loops if we want

```
for (( ; ; ))
doi
    echo "infinite loop [hit CTRL+C to stop]"
done
```
We can use `break` to exit `for`, `while` and `until` loops early.

```bash
for i in someset
do
    cmd1
    cmd2
    if (disaster-condition)
    then
        break
    fi
    cmd3
done
```
We can use continue to skip to the next iteration of a for, while or until loop.

```
for i in some set
do
cmd1
   cmd2
   if (i don’t like cmd3-condition)
      continue
   fi
   cmd3
done
```
case

case allows you to execute a sequence of if else if statements in a more concise way:

case expression in
  pattern1 )
    statements ;;
  pattern2 )
    statements ;;
  ...
esac

Here the patterns are expanded using shell expansion.
$ read -p "What is you size?" type
$ case $type in
tall)
  echo "yay tall"
  ;;
short | petite)
  echo "your height is either short or petite"
  ;;
[[[:digit:]]]?)
  echo "We do have your number"
  ;;
*)
  echo "I don’t get it :(" 
  ;;
esac

- the case statement stops the first time a pattern is matched (unless & after ;;).