# If example

```
#!/bin/bash
# 2006 09 08 - demonstrate if / then / else
if [ "x$1" != "x" ] && [ -f "$1" ]
then
   echo -n "Remove $1 (n)? "
   read answer
   if [ $answer == "y" ] || [ $answer == "Y" ] || [ $answer == "yes" ]
   then
      echo "Would remove"
   else
      echo "Would NOT remove"
   fi
else
   echo "Please specify a regular file"
fi
```

# If example

```
#!/bin/bash
# 2006 09 08 - demonstrate if / then / else
if [ "x$1" == "x" ]
then
  echo "Please specify a regular filename!"
  exit 1
elif [ ! -f "$1" ]
then
  echo "$1 is not a regular file!"
  exit 1
else
   echo -n "Remove $1 (n)? "
   read answer
   if [ $answer == "y" ] || [ $answer == "Y" ] || [ $answer == "yes" ]
   then
```

```
      echo "Would remove"
   else
      echo "Would NOT remove"
   fi
fi
```

# The case statement

```
   case WORD in PATTERN1 ) COMMANDS ;; PATTERN2 ) COMMANDS
;; ...  esac
```

The idea here is that WORD is tested against the various PATTERNs listed, in order. The first match then executes the associated COMMANDs.

# Case example

```
#!/bin/bash
# 2006 09 08 - case example
case $1 in
  "yes")
     echo "Thanks!"
     exit 0
     ;;
  "no")
     echo "Okay!"
     exit 1
     ;;
  *)
     echo "Please use either ’yes’ or ’no’ (case-sensitive)"
     ;;
esac;
```

# While/until loops

```
while list; do list; done;

until list; do list; done;
```

`while` executes the do list as long as the **last** command in the `list` returns 0. `until` executes until the last command in the `list` returns 0.

# while example

```
#!/bin/bash
# 2006 06 08 -- rdl
echo -n "Now ’finish’ ? "
read cmd
while test $cmd != "finish"
do
   rm NONEXIST
   echo "Status of \$? == $?"
   echo -n "Now ’finish’ ? "
   read cmd
done
```

# until example

```
#!/bin/bash
# 2006 06 08 -- rdl
echo -n "Now ’finish’ ? "
read cmd
until test $cmd == "finish"
do
   rm NONEXIST
   echo "Status of \$? == $?"
   echo -n "Now ’finish’ ? "
   read cmd
done
```

# Shifting the arguments

You can “shift” the argument list, eliminating the current \$1 and replacing it with the current \$2, and so forth:

# Shifting the arguments

```
#!/bin/bash
while [ $# -gt 0 ]
do
  echo "$# --> arguments == ’$@’"
  shift;
done
```

# Shifting the arguments

```
[langley@sophie 2006-Fall]$ ./Script8.sh a b c d e f g h
8 --> arguments == ’a b c d e f g h’
7 --> arguments == ’b c d e f g h’
6 --> arguments == ’c d e f g h’
5 --> arguments == ’d e f g h’
4 --> arguments == ’e f g h’
3 --> arguments == ’f g h’
2 --> arguments == ’g h’
1 --> arguments == ’h’
[langley@sophie 2006-Fall]$
```

# `exit`

We have already talked about `exit`, but to reiterate some points about exit:

☞ An `exit` status of zero should indicate success. It is a good idea to use an explicit `exit NUM` in scripts.

☞ An `exit` status that is non-zero should indicate failure.

☞ C programs use `exit(NUM)` to return a status.

# exit example

```
#/bin/bash
# 2006 09 08 -- rdl Script9.sh
if ./Script10.sh
then
  echo -n "Enter filename: "
  read filename
  echo "You entered '$filename'"
else
  echo "Okay, no filename needed."
fi
```

# exit example

```
#/bin/bash
# 2006 09 08 -- rdl Script9.sh
while /bin/true
do
  echo -n "Should I ask for a filename? "
  read answer
  case $answer in
     "no")
            exit 1
            ;;
     "yes")
            exit 0
            ;;
     *)
            ;;
  esac
done
```

# Regular expressions

Regular expressions are a convenient way to describe a sequence of characters, and regular expressions are part of such programs as `emacs`, `awk`, and `perl`.

# Regular expressions: operations

Concatenation: just place items adjacent, such `ab`, `xyz`, or `somechars`

# Regular expressions: operations

Repetition: we use “*” to indicate repetition zero or more times:

```
a*b == b, ab, aab, aaab, ...
```

# Regular expressions: operations

Special case of repetition: we can specify one or more times with $+$:

```
a+b == ab, aab, aaab, ...
```

# Regular expressions: characters and classes

The dot "." can indicate any character, such as

```
a.b == a1b, a2b, a3b, ...
```

# Regular expressions: characters and classes

To specify a class of characters, you can use the [ ] syntax:

[abc] == a, b, c

[a-d] == a, b, c, d

[â-z] == NOT a lower case character

[0-9] == 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

# Anchoring

You can “anchor” an expression to either the beginning of a string or its end, or both. Use t̂o indicate the beginning of a line, and $ to indicate the end:

`âbc$` matches a line that consists exactly of `abc`

`abc$` matches a line that ends in `abc`

`âbc` maches a lines that begins with `abc`

# Alternation and grouping

You can specify a group with round brackets "( " and ")".

You can specify alternatives with a vertical "|"

`(abc)|(def)` matches either `abc` or `def`

# Note on grouping

It also possible in many instances possible to make a reference to whatever matched a group in round brackets.

# Check chapter 32 for more on regular expressions

32.20 has a good summary of metacharacters for different programs.

32.21 has a reference with many useful examples

# Using grep/egrep

You can use the grep program to find strings in files. The “-i” option makes the search case-insensitive. If no file or files are specified, then grep looks to stdin for input. grep also adds “?” as a special character that matches 0 or 1 instance of any character.

# Examples with grep/egrep

```
egrep  [Ll]angley *   # finds instances of ‘‘langley’’ or
                      # ‘‘Langley’’ in all files in the
                      # current working directory
egrep -i she?p *      # finds case-insensitive instances of
                      # shep and she.p
egrep -c /bin/bash *  # shows filename and
                         # number of matches
```

# Popular options with `grep/egrep`

- ☞ -i → case-insensitive
- ☞ -c → display count of matching lines rather all matching lines
- ☞ -v → invert the matching
- ☞ -H → always show filenames
- ☞ -h → always suppress filenames

☞ -l → just show the filenames that have one or more matches

# wc

You can use the wc program to count characters, words, and lines:

```
wc -l *     # count the number of lines in all files
wc -w *     # count the number of words in all files
wc -c *     # count the number of characters in all files
wc -lw *    # count the number of words and lines in all files
wc *        # count words, characters, and lines in all files
```