```
#!/bin/sh

#
#  myscript.sh
#
# General purpose script for extracting Glycine
# occurrences in a datafile.
#
# Usage: myscript.sh datafile
#
# Exit values: 1: No datafile given or file
#                 doesn’t exist
#              2: No Glycine found
#
# Author: Me, myself and I
# Date:   Heidelberg, December 12., 2012
#

# --- Configuration ---
GREPCMD=/bin/grep
DATAFILE=$1

# --- Check prerequisites ---
# first check for $1
if [ -z $DATAFILE ]
then
  echo “No datafile given” 1>&2  # print on STDERR
  echo “USAGE: $0 datafile”
  exit 1
fi

# then check if the file exists
if [ ! -f $DATAFILE ]
then
  echo “Datafile $DATAFILE does not exist!” 1>&2
  exit 1
fi

# --- Now processing---
$GREPCMD –q Glycine $DATAFILE   # Where is Glycine?

# --- Exit ---
if [ $? –eq 0 ]
then
  exit 0
else
  exit 2
fi
```

Group, group, group

One after the other:	`cmd1 ; cmd2`
One or both:	`cmd1 && cmd2`
Only one of them:	`cmd1 || cmd2`
Cuddling (there):	`( cmd1 ; cmd2 )`
Cuddling (here):	`{ cmd1 ; cmd2 }`

```
if condition1
then
  statements
elif condition2
  more statements
[…]
else
  even more statements
fi
```

```
if grep -q root /etc/passwd
then
  echo root user found
else
  echo “No root???”
fi
```

Twice the same

```
if [ -e /etc/passwd ]
then
  echo /etc/passwd exists
else
  echo /etc/passwd does NOT exist
fi
```

```
if test -e /etc/passwd
then
  echo /etc/passwd exists
else
  echo /etc/passwd does NOT exist
fi
```

```
case variable in
   pattern1)
      statements_1
      ;;
   pattern2)
      statements_2
      ;;
   […]
   *)
      statements_3
      ;;
esac
```

```
case $PATH in
 */opt/* | */usr/* )
    echo /opt/ or /usr/ paths found in \$PATH
    ;;
 *)
    echo ‘/opt and /usr are not contained in
$PATH’
    ;;
esac
```

```
for variable in list
do
  statements
done
```

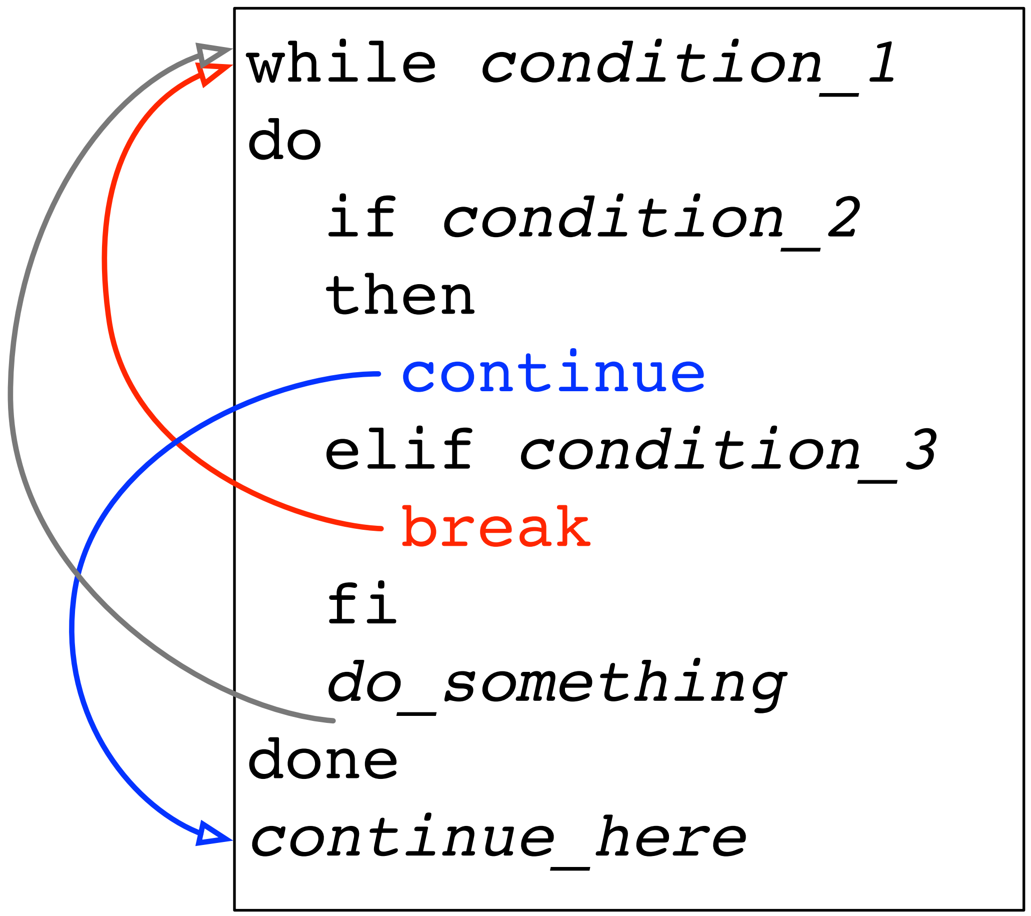
Twice the same again

```
for FILE in /tmp/*
do
  echo “ * $FILE”
done
```

```
for FILE in `ls /tmp`
do
  echo “ * $FILE”
done
```

```
while condition
do
  statements
done

until condition
do
  statements
done
```

Manual Loop Control



```
while condition_1
do
   if condition_2
   then
     continue
   elif condition_3
     break
   fi
   do_something
done
continue_here
```

Script Flexibility: Variables

Instead of

```
#!/bin/sh

echo “The directory /etc contains the following files:”
ls /etc
```

use

```
#!/bin/sh

MYDIR=/etc

echo “The directory $MYDIR contains the following
files:”
ls $MYDIR
```

Script Flexibility: Settings File

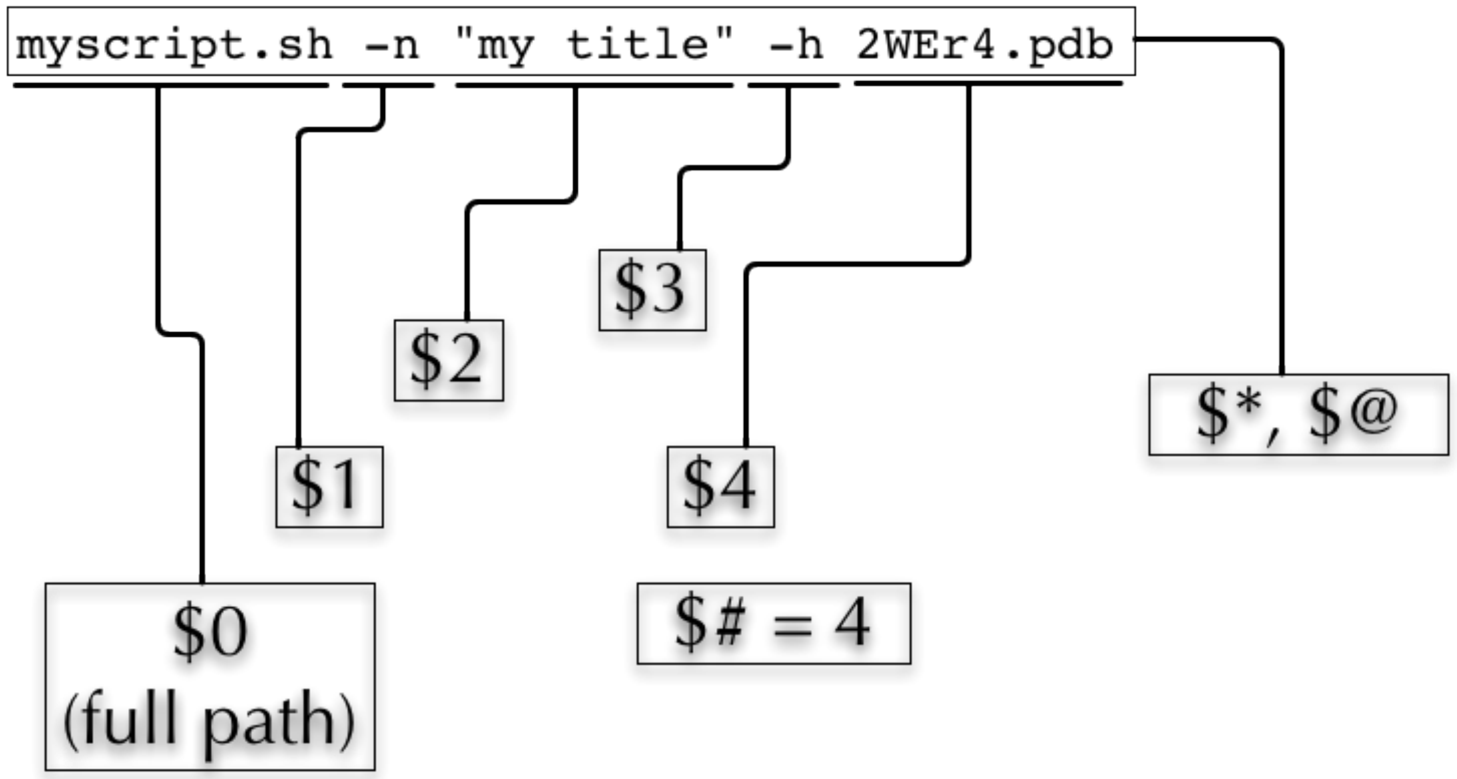
Create a settings file:

```
MYDIR=/etc
```

And source it in your script

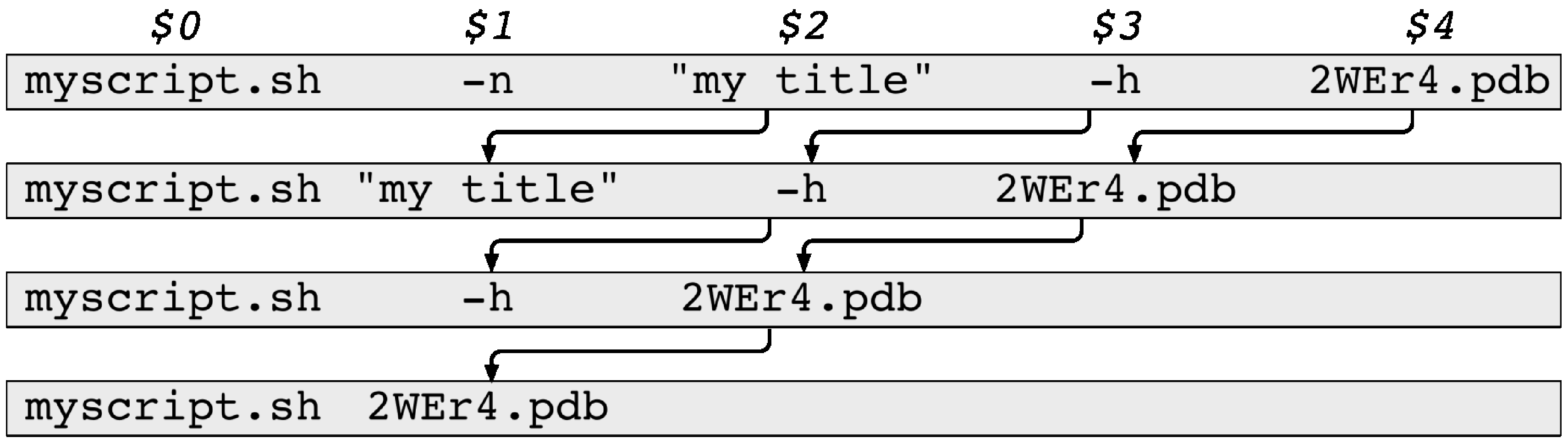
```
#!/bin/sh

. ./settings.ini

echo “The directory $MYDIR contains the following
files:”
ls $MYDIR
```

Script Flexibility: Commandline Parameters



```
"$@" = "myscript.sh" "-a" "my title" "-h" "2WEr4.pdb"
```

Script Flexibility: Walking through the Commandline Parameters


$0
$1
$2
$3
$4
myscript.sh -n "my title" -h 2WEr4.pdb
myscript.sh "my title" -h 2WEr4.pdb
myscript.sh -h 2WEr4.pdb
myscript.sh 2WEr4.pdb

Script Flexibility: Applying the case statement

```
while [ "$#" –gt 0 ]
do
  case $1 in
    -h) echo “Sorry, no help available!”  # not very helpful, is it?
        exit 1                            # exit with error
        ;;

    -v) VERBOSE=1                         # we may use $VERBOSE later
        ;;

    -f) shift
        FILE=$1                           # Aha, -f requires an
                                          # additional argument
        ;;

    *)  echo “Wrong parameter!”
        exit 1                            # exit with error
  esac
  shift
done
```

Script Flexibility: Unsolved cases regarding commandline parameters

- How to handle multiple instances of the same parameter?
- How to handle commandline arguments which are not options?

Ending a script properly:
The Exit Status

There is **always** an exit status: The exit status of the last command run in the script

The exit status of the last run command is available in the **$?** variable

Either you control the exit status or it controls you

Ending a script properly: The Exit Status – miserable failure

Ran the following scripts on the cluster

```
#!/bin/sh

[... Lots of processing steps.  One of them failed ...]

Echo “End of the script”
```

The jobs apparently failed (no result files were written) but there were no entries in the error file and the cluster administrators confirmed repeatedly, that all these scripts ran fine and successfully

WHY?

Ending a script properly: The Exit Status – good solution

This solved the situation

```
#!/bin/sh
mystatus=0;

[... do something that might fail ...]
if [ $? -ne 0 ]
then
  mystatus=1
fi

[... do something else that might fail, too ...]
[ $? -ne 0 ] && mystatus=1          # same as above.  Do you understand
                                    # this?

echo "End of the script"
exit $mystatus
```

The exit status had controlled us, but now **we** are back in control