Writing a simple C-shell script

Archive:

- `Scripts/` link `http://msekce.karlin.mff.cuni.cz/~dolejsi/Vyuka/NS_source/cs/Scripts.tgz`

Contents of the directory

```
-rw-r--r--  1 dolejsi dolejsi  641 úno 19  2019 Makefile
-rwxr--r--  1 dolejsi dolejsi   24 úno 19  2019 mk1.sh*
-rwxr--r--  1 dolejsi dolejsi  385 bře  5 08:41 mk2.sh*
-rw-r--r--  1 dolejsi dolejsi 2243 úno 19  2019 NC.f90
-rw-r--r--  1 dolejsi dolejsi 2524 úno 19  2019 NC2.f90
```

- basic script **`mk1.sh`** allows a translation nad running of the code by one command
- file `NC.f90` is from **`tutorial5`** and **`NC2.f90`** is its small modification
- compare both files by **`meld NC.f90 NC2.f90 &`**
- `NC2.f90` allows to read the data from an external file, syntax is
  `./NC2.f90 data.dat`
- `NC2.f90` carries out computation only for one level of partition
- number of refinement levels can be driven by the script **`mk2.sh`**, see below

The ”simplest” C-shell: `mk1.sh`

```
#!/bin/csh

make
./NC
```

- 1st line is the notation, that C-shell will be used
- it only translate and run the old code `NC.f90` (nothing more)

---

more about C-shell: link `https://www.dur.ac.uk/resources/its/info/guides/3Cshells.pdf`

The more advanced C-shell: `mk2.sh`

```
#!/bin/csh
if (  $#argv != 2) then
 echo ’Syntax: mk2.sh <# levels> <output_file>’
 exit
endif

make

@ max_lev = $argv[1]
set output_file = $argv[2]
rm -f $output_file

@ i = 1
while($i < $max_lev)

cat << EOF > data.dat
 $i
EOF

./NC2  data.dat > out2

tail -n 1 out2 >> $output_file

@ i++
end

echo "Computation has been finished"
echo "Output is in file ’"$output_file"’"
```

Syntax, e.g., : `./mk2.sh 5 results`

- study the script line by line by reading and running the code
- command `./mk2.sh 5 results` runs the computation of the integral starting from level 1 to level 5. Modify the code `NC2.f90` and script `mk2.sh` such that it runs from `lev_min` to `lev_max`, i.e., for example `./mk2.sh 2 5 results`
- the output file `results` of `./mk2.sh 5 results` is suitable for gnuplot, visualize it by

  ```
  gnuplot> p 'results' u 1:3 w lp,'results' u 1:4 w lp,'results' u 1:5 w lp
  ```

  what this figure means.
- Similarly, visualize the error of the computation:

  ```
     gnuplot> set logscale
     gnuplot> p 'results' u 1:6 w lp,'results' u 1:7 w lp,'results' u 1:8 w lp
  ```

- Modify the script `mk2.sh` such that it automatically create the figures as eps files. Minimal example added to the end of `mk2.sh`:

  ```
  cat << EOF > tisk.gnu
  set terminal postscript eps color
  set colorsequence classic
  set output "figs1.eps"
  p '$output_file' u 1:3 w lp,'$output_file' u 1:4 w lp,'$output_file' u 1:5 w lp
  set logscale
  set format "%2.0e"
  set output "figs2.eps"
  p '$output_file' u 1:6 w lp,'$output_file' u 1:7 w lp,'$output_file' u 1:8 w lp
  EOF

  gnuplot tisk.gnu
  echo "Figure files 'figs1.eps' and 'figs2.eps' created"
  ls -l figs1.eps figs2.eps
  ```

- figures can be visualised, e.g., by `evince figs1.eps &` or `okular figs1.eps &`