Name:_____

Environment

Environment is a set of variables handled by the shell, that contain mostly user settings. The exercises here use bash shell. Other shells have slightly different syntax.

1. Print your environment: *printenv*
2. Check your path, it's found in variable *PATH*: *printenv PATH*
   Path is a list of directories that the shell will use when trying to find executable commands.
3. Let's make a new variable: *export ESIM=helloworld*
4. You can then use the variable on command line: *echo $ESIM* or *echo ${ESIM}*
5. Variable can also be given for one command line only: *VALIAIK="heipa hei" echo $VALIAIK*
6. Make a directory *bin* to your home directory and add it to your path: *export PATH=${PATH}:~/bin*

(On Debian, if you have the bin directory, it will be added to your path on next login.)

Script files

Script file (or batch file) is an executable file that contains a list of commands that the shell will execute consequentially. Script files are identified by adding a so-called hashbang identifier (*#!/bin/sh*) to the beginning of the file. The identifier tells which interpreter to use when executing the file. The file also has to be marked executable: (*chmod +x file*). Sometimes file name extension *.sh* is used, but that's not mandatory.
Script files have their own limited programming language, but their usability in automating tasks is undeniable. Here we'll demonstrate the structure with few simple examples.

1. Creat a file named *my_programs.sh* to your bin directory and enter following commands there:

   ```
   #!/bin/sh
   # Hashmark denotes comments in script files
   # Save output of one command to a variable:
   MINAITSE=`whoami`
   # Another way to do the same thing:
   # MINAITSE=$(whoami)
   # Print out the list of your processes and count the number of lines
   ps --no-headers -u $MINAITSE | wc -l
   ```

   Here the character used around the whoami command is backtick. On finnish keyboards it's found between plus and backspace keys using: *shift-‘ space*. You have to be careful with these.

2. Add executable permission: *chmod +x my_programs.sh*
3. Now you can run the command: *my_programs.sh* and it will print out the number of processes you have currently running.

Control structures

Script files have control structures (syntax depends on the shell; bash/tcsh/zsh) like if/then/else trees and for loops. Bash manual is frustratingly verbose (5700 lines?) but for this topic I suggest you read the chapter named "CONDITIONAL EXPRESSIONS".

1. Let's start a new script file. Add the following commands:

```
#!/bin/sh
# Check that we won't run the command as root
if [ `whoami` = 'root' ]
then
    echo 'Do NOT run as root, dummy!'
    exit 1
else
    echo "Welcome, `whoami`."
    # Note: Difference between double (") and single (') quotes is that
    # stuff between double quotes is also interpreted.  Try switching double
    # quotes to single quotes in the previous command.
fi
```

2. Test the program as normal user and as root user (for example, using sudo command).

3. Often the same command has to be run multiple times and output is to be written to different files. Let's make an example of a for loop and *seq* command. Add to the previous file:

```
for i in `seq -w 1 10`
do
    date > timestamp${i}.txt
    echo Sleeping $i seconds.
    sleep $i
done
```

4. Thus the syntax is *for VARIABLE in VALUES.. ; do commands ; done*. Let's continue:

```
for each in timestamp*.txt ; do
    cat $each
    rm $each
done
echo All done.
```

Command parameters

Script files can take parameters as they are called from command line. These can be referred to with $NUMBER . Zero is the name of the executed file and the parameters follow up on higher numbers. For example:

```
#!/bin/sh
echo This script is `basename $0`
# Is there a parameter?
if [ "$1" = "" ] ; then
    echo Master, give me some parameters
    exit 0
fi
if [ -r "$1" ] ; then
    echo First parameter is $1 : `wc -l "$1"` lines
else
    echo There\'s no such file as $1
fi
```