# Operating Systems Assignment 0: Basic Usage of Command-Line Operating Systems

February 2, 2011

## 1 Introduction

During class we have seen how exactly shells work. The shell usually is the interactive user interface to the operating system[1]. The main purpose of the shell is to start and execute commands. In class we have also gone over several advanced features of the "bash" shell, several useful utilities found on UNIX-like operating systems and how to create and execute script files.

In this practical class, we will put the things we have seen today into practice to acquire some hands-on experience. We hope that you can put this hands-on experience to use in the upcoming assignments of this course and in other courses as well in order to save time by automating things. For the upcoming lab assignments, we assume that you have working knowledge of the shell's fundamentals.

*Note: we will be working on the assignments in room 411, but when working on silver (ssh.liacs.nl) make sure to be using "bash" and not "tcsh".*

## 2 Pipes and redirection

We have discussed the ability to create pipelines and redirect file descriptors to other files or running programs.

**Exercise 1.** Create a file with the following contents, or any other file with at least two words per line:

```
one two
three four
five six
seven eight
nine ten
```

Create a command that removes the first word from each line and sorts the result. (Hint: look at the `-f` option of `cut`).

**Exercise 2.** Create a pipeline which will remove the line starting with `five` from the output and sorts the result. (Hint: use the `grep` manual to find an appropriate option to use).

**Exercise 3.** If we use the command `cat` with two arguments, the filename of an existing file (let's say the file is named `numbers`) and the filename of a non-existing file, we will receive as output the contents of the existing file as well as an error for the second file. When we pipe the

[1]However, many embedded control systems do not have such an interface

results of the `cat` command to `less`, the `less`-output will only contain the contents of the existing file. Modify the command line:

```
cat existing acdefg | less
```

such that both the file contents and the error appear in the `less`-output.

**Exercise 4.** Same scenario as exercise 3, but now modify the command line such that the file's contents and the error are directed into two separate files.

**Exercise 5.** Same scenario as exercise 3, but now modify the command line such that all output is directed to a single file.

## 3 Shell scripting

**Exercise 6.** Create a shell script which is a wrapper around the `ls` command and passes the `-al` switches by default together with the arguments already specified on the command line.

**Exercise 7.** Create a file with the following contents:

```
/bin/test
/usr/bin/test
/usr/bin/thisisfake
/usr/bin/mail
```

and write a shell script that will check whether these files exist. The script should output a string saying whether or not the file exists.

**Exercise 8.** On UNIX systems, the directory `/usr/bin` usually contains a lot of executable programs. Write a script which will output all programs with a filename of length 8 or longer. The shell utility `expr` can help you to determine the length of a string.

**Exercise 9.** Often a shell script is used to start and stop a "daemon" process. A daemon process is typically a server (for example a webserver or mailserver) which runs in the background. These scripts take a parameter in the form of a verb, such as *start* or *stop*. As a process to run in the background, we will use `xterm`. Firstly, become familiar with the different job control commands, such as appending `&` and using the `fg`, `bg`, `jobs -l` and `kill` commands.

Secondly, write a script that can take the `start` and `stop` verbs as parameter to start and stop an xterm. Use a `case` statement to identify the verb.

Some more hints:

1. The special variable `$!` will return to you the pid of the last started process.
2. Write out the pid of the process you have started to a file. This is what is also typically done by the Linux daemon start/stop scripts.
3. You might want to use functions in bash:

```
function my_function()
{
    echo "Hello"
}

echo "Outside function"
my_function
```

4. When the basics are working, make the script fool proof. Do not start duplicate processes, print an error and fail if there is no currently running process.

Do not hesitate to ask questions to the assistants!