**Unix and C Programming 04**

Your working environment is defined whenever you login or start another shell. This environment is set using the values that the shell finds in initialization files which it always reads as it starts up. Depending on the shell you are using, different files are read. In our case we are using the ”Bourne Again” shell “bash”, which is an extension of the Bourne shell “sh”. The environment in our case is setup through a few global files /etc/profile and /etc/bashrc, which you cannot edit, and a file in your home directory named .bashrc. Note that all files starting with a single dot are hidden, i.e. are not shown when you “ls” your files. You will need to use “ls –a” to list *all* files in a directory (do it now).

**Environmental Variables**

Some variables that are defined by default are, for example USER, SHELL, PATH, HOSTNAME, TERM, PS1, and PROMPT\_COMMAND. The most important of these variables is the PATH. It contains a list of directories that your shell will look through to find the program you want to execute. For example, if your PATH is “/usr/local/bin:/bin:/usr/bin” and you type ls at the command line, the shell searches through the directories /usr/local/bin, /bin, and /usr/bin, in this order, for a program (i.e. a file marked “executable”) named ls. If it finds it, it executes it, otherwise you’ll see an error message (which directory *does* contain the ls program)? The PS1 variable defines how your prompt looks like, and the PROMPT\_COMMAND defines the command that executes as part of your prompt.

To show the value of a variable, use, for example, the echo command and preface a variable by a dollar sign $:

 echo “I am $USER logged in at $HOSTNAME”

The command set will list all defined variables; to set a variable, use for example NAME=”Bert Wachsmuth”, followed by export NAME if other programs are supposed to know about your new variable.

You can edit your .bashrc file to define new variables or to redefine existing ones permanently, because that file is read whenever you login. To execute that file without login in again, use source .bashrc

For example, check if there is a variable BOSS defined via echo $BOSS. You should see an empty line. Now edit the file .bashrc and add the lines

BOSS=”Bert Wachsmuth”
export BOSS

Just because you added these lines does not yet define the variable. But now type source .bashrc then check the value of BOSS again. It should be defined now. Also logout and login, then check for the BOSS again. It should be a defined variable from now on.

**Setting up Aliases**

In addition to defining variables in .bashrc you can also define *aliases*, i.e. commands that are a shortcut for another command. You can for example define the command dir as a shortcut for “ls –la | more”. The syntax for defining an alias is alias new\_command=”old\_command”, so to define the new dir command you would say alias dir=”ls –la | more”. You could either define this alias on the command line, so that it works for your current session, or you can add that to your .bashrc command so that it gets defined every time you login.

Finally, you can add regular commands to .bashrc that will execute once every time you login. For example, add the following line to the end of that file:

echo “Welcome $USER. You are using $HOSTNAME”

You can even embed the output of a program into a string printed out by echo, but you need to surround the program in “reverse single quotes”. For example, the date command prints out today’s date and time. If you say:

echo “Welcome $USER. You are using $HOSTNAME. It is date”

it would simply print out the word date, but if you used reverse quotes:

echo “Welcome $USER. You are using $HOSTNAME. It is `date`”

then it will insert the current date and time. This is properly called ‘command substitution’. Try to use the “who” command instead of $USER to insert your name into the echo string above.

**Shell Scripts**

As well as using the shell to run commands you can use its built-in programming language to write your own commands or programs. You can put commands into a file - known as a shell script - and then execute that file as you would a command or program. The steps are like this:

* Create a text file containing your program, i.e. list of commands, and save it. Any name will do, but try to make it descriptive.
* Try out your script by executing it via “sh name”. Using sh ensures that you will use a copy of the Bourne shell to execute it so your program needs to comply with any syntax rules of that shell. Another way to ensure this is to use as the first line of your script the line

 #!/bin/sh
* When your script works to your liking, you can mark it as “executable” via chmod u+x name. You could also allow others to execute it by setting permissions accordingly. After marking your script executable you can run it by simply typing the script name, or, depending on your path setting, by typing ./name. Finally, you could move your script into your bin directory, which by default is part of your path (verify), so that you can execute your script from any directory by just typing its name.

For example, start editing a new file using pico myinfo. Add the following lines to the file:

#!/bin/sh

# Author: Bert Wachsmuth

# Date: 2/2/2011

#

# This script displays the date, time, userinfo and current directory

echo "Date and time is: \c"

date

echo

echo –n "Your username is: `whoami` \n"

echo –n "Your home directory is: $HOME \n"

echo –n "Your current directory is: \c"

pwd

Notes

* the first line should always be present for your shell scripts
* lines starting with a hash mark (other than the one in the first line) indicate that the line is a comment, not a program or command to execute. You should always use plenty of comments to explain your script’s function, even if it is clear to you now.
* The command whoami is in reverse quotes, which is why it executes
* There are certain backslash characters, which have special formatting meaning

After saving the file as myinfo, you can execute it by typing

 sh myinfo

If you satisfied, mark the file executable by saying chmod u+x myinfo. Now you can execute it by typing

 ./myinfo

Finally, make sure you have a directory bin and move that file into your bin directory. Now you can type myinfo from any directory to execute your script.

**Questions**

1. What other ‘hidden’ files, if any, are in your home directory?
2. Which directory does contain the ls program?
3. What is the value of your PS1 variable?
4. Make sure to define your name in the variable USERNAME. Add it to your .bashrc file and verify that it worked using echo “My name is $USERNAME”.
5. What is the value of your PATH variable? Edit it so that “/home/yourname/prgs” is *appended* to your PATH (but be sure not to lose your existing path). Be careful, if you reset your path incorrectly, nothing will work anymore (but it can be fixed, of course)
6. Add a new alias dir to your .bashrc to list a directory in a way you find convenient
7. Redefine the “mv”, “cp”, and “rm” commands so that they are safe to use, i.e. they will prompt you before overwriting an existing file. Use the “man” command to find the right switches. For example, using man mv you will find that the –i switch will “prompt before overwriting”, so the appropriate alias would be alias mv=”mv -i". Be sure to similarly redefine rm and cp
8. Use the man command to get more info about the echo command, in particular to find out what the –e switch does.
9. Create the script myinfo as outlined above and verify that it works properly.
10. As we know, the grep command searches for occurrence of a pattern. As the man command shows (verify), the –c flag cause grep to display the number of occurrences of a pattern. Use the last command (which lists when everyone who logged in this month), a pipe, the grep command, the USER variable, and the –c switch to create a shell script that displays how many times you have logged in this month. Display the answer in a nice sentence. Move your working script to your bin directory and mark it executable. Why is it better to use USER instead of hard-coding your username into the script?