Lab 2: Shell, Environment Variables, Useful C-Shell Commands

1 Introduction

- main topics:
  - what is shell, different shells,
  - bourne shell, c-shell,
  - path, environment variables,
  - other useful commands.
- Useful links:

  A good reference to UNIX providing basic facts from history, introduction to its structure, shells and commands is at
  http://wks.uts.ohio-state.edu/unix_course/

  Other pages discussing different shells are:

  - shell in general, bourne shell:
    http://www.cc.vt.edu/cc/us/docs/unix/shells.html
    http://www.ling.helsinki.fi/users/reriksso/unix/shell.html
  - korn shell
    http://www.kornshell.com/info/
  - why not cshell:
    http://www.cs.ruu.nl/wais/html/na-dir/unix-faq/shell/csh-whynot.html

2 UNIX, Kernel, Shell.

In UNIX, communication with the operating system is not direct. Instead, UNIX commands are interpreted by a program called shell. Shell translates the commands into actions that are taken by the operating system, the kernel. Kernel interacts directly with the hardware and provides services to the user programs.

As mentioned above, **Shell is a program that allows the system to understand user's commands** (a command interpreter) Shell is analogous to command.com in DOS.

2.1 Purpose of Shell

- **interactive use:** system waits for the command at the UNIX prompt and interprets it.
- **customization of your UNIX session:** shell defines variables controlling the behavior of user's UNIX session.
- **programming:** shell allows to create programs called shell scripts.

2.2 Different Shells: which, when, why

- *sh*: the Bourne shell
  - developed by Steven Bourne at AT&T Bell Laboratories
  - most compact but also simplest shell
  - very good features for controlling input and output, but not well suited for an interactive user
  - good choice for writing shell scripts
  - default prompt is $
- *csh*: the C shell
  - developed by Bill Joy at the University of California
  - uses C type syntax
  - input/output implementation is worse than in Bourne shell which makes it not very good for programming.
  - has a job control allowing to reattach a job running in the background to the foreground.
  - provides a history feature allowing modification and repetition of previously executed commands
  - is suitable for interactive use.
  - the default prompt is %.
- *ksh*: the Korn shell
  - designed and developed by David G. Korn at AT&T Bell Laboratories
  - contains the best features of sh and csh, plus many new features of its own
- *bash*: the "bourne again" shell

  - originally written by Brian Fox of the Free Software Foundation, current developer and maintainer is Chet Ramey of Case Western Reserve University
  - contains a number of enhancements over bourne shell, both for interactive use and shell programming

- *tcsh*: the extended version of c shell
  - is suitable for interactive use
  - has the same features as c shell plus several additional user friendly features: use of arrows to repeat previous commands, the first letters can be completed into a unique file name by hitting the tabulator.

By default, U. of Waterloo user UNIX account is set up with c-shell, though other shells are also available. For interactive use you can choose any shell you wish. However, our labs will refer to c-shell (*csh*) and tc-shell (*tcsh*) respectively for interactive use. For programming purpose we will always use bourne shell (*sh*).

2.3 Changing your shell

To check what shell you are using right now, type:

grep $USER /etc/passwd

To change your shell, type

chsh

chsh shows first the current shell and then prompts for the new shell. Less experienced users may choose *tcsh* instead. The shell will be changed permanently in few hours after typing the *tcsh*. For a temporary shell change type the name of the shell on the command prompt.

For example, type

sh

to work in bourne shell immediately. The *sh* prompt,$, appears and you can enter *sh* commands, say,

ls
cd cs241
mkdir lab2
pwd

Once finished, you may end the work and return to the previous shell using *CTRL-D.*

Notice that most UNIX commands introduced till now are valid in all shells. From now on, when discussing interactive use, we focus on c-shell or tc-shell only. The valid syntax for korn shell or bourne shell can be found in UNIX in a nutshell book.

Before discussing internal and external commands, we need to understand the structure of the filesystem in UNIX.

2.4 The Tree Structure of the Filesystem and a Home Directory

Files are organized into directories. A directory is a special kind of file that list other files. A directory can contain any number of files and other directories. The directory at the top is called the "root" and has a special name "/" ("slash")
(picture)

Log in leads in the directory created for you by the system administrator. This directory is called your home directory. For example cs241 home directory is:
/u/cs241.
/ is the root and 'u' stores home directories for all users. The current directory can be changed to your home directory by typing
cd
with no pathname.

2.4.1 Current Directory, Absolute and Relative Pathname

Pathnames locate files in the UNIX filesystem. By default, UNIX looks for files or directories starting from the current directory. For example, if the current directory is /u/youruserid/cs241 then the UNIX command

a) *cd a1*

will switch into /u/youruserid/cs241/a1, if such a directory exists. Otherwise it will print an error message. If /u/youruserid/cs241 is the current directory then typing

b) *more a1/q1.txt*

you ask UNIX to locate the file q1.txt within the directory /u/youruserid/cs241/a1. You can also type:

c) *more /u/youruserid/cs241/a1/q1.txt*

to view the same file.
Cases a) and b) use relative pathnames, case c) an absolute pathname.
Several abbreviations simplifying the work with pathnames exist:

. refers to the current directory
.. refers to the parent directory

For example, if /u/myuserid/cs241/a1 is the working directory and q1.txt is the file in a1 directory, then

more q1.txt

has the same effect as

more ./q1.txt

If the working directory is the same as above but you want to check the file q0.txt in directory a0 without changing the current directory, type

more ../a0/q0.txt .

In C shell, ~ is a shortcut to the home directory. For example, ~/cs241/lab2 is equivalent to the absolute pathname */u/myuserid/cs241/lab2*. *~username* switches to the home directory corresponding to username.
It this sense, UNIX allows practically anything and it the user's responsibility to set up permissions on his/her files in a way that the above features cannot be misused.

In summary,

- the absolute pathname always starts with /
- the C shell turns ~ into an absolute pathname starting at the home directory or at *~username* directory
- if the pathname does not begin with / or ~ (C shell) then the pathname is relative to the current directory.

As we will see shortly, it is very important to understand the difference between absolute and relative pathname.

2.5 Internal and External Commands

Internal commands, such as *cd*, are build into the shell. The shell interprets the cd command and changes your current directory accordingly. External commands are just external programs. Example of an external command is *ls*, which is stored in the file /bin/ls.

A build-in command or a command containing an absolute pathname starting with /, is executed by shell directly. Say, /bin/ls executes the program called *ls* located in the directory bin. If the command is neither build-in, nor specified by an absolute pathname, shell looks in its search path first.

2.5.1 PATH

The search path is set up in the environment variable *PATH*. To see the content of PATH, i.e. all pathnames available to shell, type *echo $*PATH

or

printenv PATH

After typing the above command, the search path will look like:

/bin:/usr/bin:/.software/local/.admin/bins/bin: usr/openwin/bin

where ':' distinguishes between different pathnames. As mentioned before, the shell seeks the program name in directories set in *PATH*. The *PATH* is searched sequentially. For example, if *PATH*=/bin:/usr/bin:/... shell starts to look for the program in /bin, then /usr/bin etc. At the moment the program is found, it is executed. Presence of the /bin directory in the *PATH* is essential, for all external UNIX commands are located there. By default, the current directory is not in the *PATH*. It can be added to the *PATH*, but it is not recommended. Why?

2.5.2 Changing PATH

Very often it will be necessary to add new directories into *PATH*. The command for setting or changing the environment variable is setenv. To add, for example, user's cs241/a1 directory to the *PATH*, type: setenv *PATH* $HOME/cs241/a1:$*PATH* This will add cs241/a1 directory to the beginning, which means that the cs241/a1 directory will be searched by shell first. Similarly one can add a directory to the end of the list. Note, that *HOME* is another environment variable, which defines user's home directory.

3 A Simple Shell Script

Scripting languages are similar to batch files in DOS environment. A shell script is a sequence of UNIX commands in a text (ASCII) file. The following sequence of commands creates a script file named first with one UNIX command:

cd˜/cs241/lab2
vi first
man -k shell | sort -u | more

(save and exit the file by pressing *ESC :wq*)

To execute it in bourne shell use:

sh first

For shell, *first* is not an executable program yet. To become one, the command

chmod a+x first must be executed. Now,

./first

executes the shell script as desired.
More complicated shell programs are discussed in lab3.

4 Accessing the Files and Security Issues.

The chmod command we used previously changes the access mode into a file. It is used to change the permission for the owner and other users to read, modify or execute the file. For example, if
ls -l

displays

−rw−r−−−−− 1 userid cs241 82 Jun 30 17:11 temp
−rw−r−−−−− 1 userid cs241 304 Jul 14 10:48 first
drwxr−x−−x 2 userid cs241 4096 May 15 17:23 labs

then the information about the files and their permissions is in the first column.

Each line of the first column starts either with '*d*' or '−'. '*d*' means that the file is a directory, '−' denotes a regular file.

The next three characters describe the permission to the owner of the file. '*r*' means file is readable, '*w*' file can be modified (i.e. it is writable), '*x*', file is executable. Symbol '−' instead of '*x*' means that the user has no permission to execute the file, similarly for reading and modifying. The next three characters, starting from fifth position, give the permission to a group including the owner. Membership of userid in the group cs241 is specified in columns three and four of the list. The last three characters of each line in column one describe the permission to all other users.
For example to allow others to read your *first* script use:

chmod o+r first

To remove that permission type:
chmod o−r first.

5 Wildcards

Wildcards may by used to specify a group of files. Here is a list of some most commonly used wildcards:

- ? (question mark) matches any single character
  example: To list all dlx files needed for assignment 1 use:

  ls a1q?.dlx

- * (asterisk) matches zero or more characters
  be careful, . has no special meaning (except for ls command) and so,
  *rm **
  will remove everything without any warning or asking however, '.' is considered special for ls command:
  Example:

  ls a.txt*

- [character-list] matches any single character that appears in the list.
  Example:
  [ab] means either a or b
  [a-z] means any character between a and z, inclusive
  ls test[12345].txt
  or
  ls test[1-5].txt

  NOTE: you will hear about regular expressions (REs) and their rules soon in class, in assignments and in lab3. Be aware, that *,?, [] have a different meaning in shell and in rules used by RE. You will need to know and use both.

6 C Shell Quoting

It is important to know the meaning of different quotation characters in c shell.

- *single quotes:* ' −> turn off special meaning of all characters until the next single quote is found.
  Example:

echo $PATH
echo '$PATH'
echo Hey, what's next? Mike's #1 friend has $value.
(output: Hey, whats next? Mikes)
Everything after # is ignored, since # starts comment in shell

- *double quotes: "* –> work almost like single quotes, but double quoting allows the char
  $ (dollar sign),
  ` (back-quote),
  \ (backslash)
  to keep their special meaning.
  Example:

  echo "Hey, what's next? Mike's #1 friend has $value."

- *pair of back-quotes:* ` does command substitution
  Example: How to send an e-mail message to all the users logged on to the system.
  First use the command
  who | cut -c1-8
  which will list all users currently log in.
  who | cut -c1-8 | sort -u
  will sort them in alphabetical order and will only display unique names
  mail ` who | cut -c1-8 | sort -u`
  will send e-mail to all of them.
  If you are not sure, how this command works (or you really do not want to send an e-mail to everybody) replace mail command with echo command:

  echo ` who | cut -c1-8 | sort -u`.

7 Standard Input and Output

7.1 Standart Input

Standart input is the source of information for a command. This is assumed to be the keyboard unless input is redirected or piped from a file or another command.

7.1.1 Redirecting Standard Input

To redirect the standard input for a command use the < (less than) character followed by the name of the input file.

For example:

mail bill < memo

This redirects the standard input to the mail command so that it comes from the file memo. The effect of this is to mail the contents of this file to the user bill.

7.2 Standard Output

Standart output is the destination for information from a command. This is assumed to be the terminal display unless output is redirected or piped to a file or another command.

To redirect the standard output from a command use the > (greater than) symbol followed by the name of the output file. If the file that you redirect standard output to does not already exist it will be created.
For example:

grep Smith /etc/passwd > popular

This redirects the standard output from the grep command so that it goes to the file popular. This file will contain all occurrences of the string Smith found in the /etc/passwd file.
Note: Redirecting standard output to a file that already exists overwrites its contents with the standard output.
You can append output to an existing file.

To append the standard output from a command to a file use >> (two greater than) symbols followed by the file name. If the file does not exist it is created. For example:

cat part1 > chapt2
cat part2 >> chapt2

The first line creates a file called chapt2 with the same contents as part1. The second reads the contents of part2 and appends them to the file chapt2. The file chapt2 contains now the data from part1 followed by the data from part2.

8 Other Useful C Shell Commands

- *history*
  displays the list of previously used commands.
  Type

history

to see last 100 commands used. For example, to repeat command 123 use:

!123

to repeat the previous command use:

!!

For other options see UNIX in a nutshell book.

- *alias*
  will assign a name to the command. For example, if one wants to use dir instead of ls then he/she must create an alias:

  alias dir ls # but please, do learn UNIX

- job control

  CTRL-Z

  stops the execution of the program

  jobs

  lists all running or stopped jobs

  kill ID

  terminates specific process ID

  Example:

  Suppose we have a program ./myprogram running. To stop it type

  CTRL-Z

  To see what programs are running or stopped type

  jobs

If you will see

[1] myprogram stopped

you can kill this process by

kill %1

- *finger users*

  displays data about one or more users
  Example:
  finger youruserid
  finger cs241

  To change the information about yourself visible to everybody use:

  chfn

  and follow the instructions on the screen.

As mentioned above, the information can be viewed by anyone. Be thus sure you only include the kind of information everybody can know.

You may also include additional information stored in the file .plan in your home directory.
To create or change your .plan file type:
cd # this will switch into your home directory
vi .plan # opens the file
Hello (some message)

After saving the file try

finger youruserid

You will see the message: *plan exists, but it is unreadable* (on the screen) because the file .plan is not world readable. To change the permission type:

chmod a+r .plan

Where 'a' stands for "all" = user + group + other as explained earlier.