Unix Shell Environments

Computer Science Department
CS2211a: Software Tools & Systems Programming
Fall 2013
Instructor: Mahmoud R. El-Sakka
Office: MC-419
Email: elsakka@csd.uwo.ca
Phone: 519-661-2111 x86996

Introduction

- What does the term "*environment*" mean?
  (from the *general Science*, not the Computer Science, point of view)?
  - An *environment* is a general term which *describes the surroundings* of a given object.
  - Such *description of the surroundings* can be carried out through
    - *A set of variables* to record the current status of various aspects
    - *Values* for these variables
  - *Local environment* (e.g., local room temperature)
  - *Global environment* (e.g., outside temperature)
- In Computer Science, same definition is valid.

Shell Variables

- Shell environment
  - Consists of a set of variables with values
  - These values are important information for
    - the *shell* and
    - the *programs* run from the shell
  - Example:

HOME	the full path name of your home directory
SHELL	the name of your login shell
PWD	the full path of the current working directory
USER	the user name of the logged-in user
TERM	the kind of terminal the user is using
DISPLAY	which X server to display on
PATH	the directories the shell should search to find a command
HOST	the name of the computer you are using
REMOTEHOST	the name of the host logged-in from
MANPATH	the directories the `man` command should search to find man pages

Shell Variables

- Shell variable names ***must*** only be made up of
  - Alphabetic characters,
  - Digits, or
  - Underscores
- Shell variable names ***cannot*** begin with a digit
- Shell variables can store either ***string of characters***, or ***NULL***
- Shell variables can be accessed by putting a $ in front of their names, e.g., `echo $HOME`
- New variables can be defined
- The values of existing variables can be changed
- ***Note that: variable names are case sensitive***

Shell Variables

- Many shell variables are defined in
  - .cshrc and .login (C-Style shells)
  - .profile (Bourne-Style shells)
- There are two kinds of shell variables:
  - ***Environment*** variables (***GLOBAL Variables***)
    - Affecting the current shell and the programs invoked from the shell
  - ***Regular*** shell variables (***LOCAL Variables***)
    - Affecting the current shell, but ***not*** the programs invoked from the shell

Dealing with Shell Variables (C-Style shells)

- Setting regular (local) variables:
  - set varname=varvalue
- Clearing regular (local) variables:
  - unset varname
- set without any argument lists ***all*** existing regular shell variables
- Example:

```
obelix[19]% set my_feeling="I like Unix"
obelix[20]% echo $my_feeling
I like Unix
obelix[21]% echo my_feeling
my_feeling
obelix[22]%
obelix[22]% unset my_feeling
obelix[23]% echo $my_feeling
my_feeling : undefined variable
```

Dealing with Shell Variables (C-Style shells)

- Setting environment variables:
  - setenv EnvironmentVariable EnvironmentValue
  - *Take care, there is NO "=" sign here!!*
- Clearing environment variables:
  - unsetenv EnvironmentVariable
- setenv without any argument, or just env, lists *all* existing environment shell variables
- Example:

```
obelix[24]% setenv my_environment "nice and clean"
obelix[25]% echo $my_environment
nice and clean
obelix[26]% set my_feeling = "I like Unix"
obelix[27]% echo $my_feeling
I like Unix
obelix[28]%
```

Dealing with Shell Variables (C-Style shells)

```
obelix[28]% echo $my_environment
nice and clean
obelix[29]% echo $my_feeling
I like Unix
obelix[30]% tcsh
obelix[11]% echo $my_environment
nice and clean
obelix[12]% echo $my_feeling
my_feeling: Undefined variable.
obelix[13]% exit
obelix[31]% echo $my_feeling
I like Unix
obelix[32]%
```

tcsh is a program invoked from the shell

Exit from the program

Dealing with Shell Variables (Bourne-Style shells)

- Setting regular (local) variables:
  - varname=varvalue
- Clearing regular (local) variables:
  - unset varname
- set without any argument lists *all* existing shell variables

There is no set here

There is no space before or after the "=" sign

Same as C-Style shell

Dealing with Shell Variables (Bourne-Style shells)

- Setting environment variables:
  - EnvironmentVariable=EnvironmentValue
  - export EnvironmentVariable
- Clearing environment variables:
  - unset EnvironmentVariable
- set without any argument lists *all* existing shell variables
- env lists ***all exported*** shell variables

- Once a variable is exported, the only way to inexpert it is to unset the variable

Same as regular variable

There is no setenv here

Only unset

Same as regular variable

Dealing with Shell Variables

- C-Style shells
  - To set a regular (local) variable:
    - `set varname=varvalue`
  - To clear a regular (local) variable:
    - `unset varname`
  - To set an environment variable:
    - `setenv EnvironmentVariable  EnvironmentValue`
    - ***Take care, there is NO "=" sign here!!***
  - To clear an environment variable:
    - `unsetenv EnvironmentVariable`
- Bourne-Style shells
  - To set a regular (local) variable:
    - `varname=varvalue`
    - ***Take care, there is NO space before or after the "=" sign!!***
  - To clear a regular (local) variable:
    - `unset varname`
  - To set an environment variable:
    - `EnvironmentVariable=EnvironmentValue`
    - ***Take care, there is NO space before or after the "=" sign!!***
    - `export EnvironmentVariable`
  - To clear an environment variable:
    - `unset EnvironmentVariable`

set, setenv, and unsetenv only used in C-Style shells

No "=" sign here

There is no space before or after the "=" sign

export only used in Bourne-style shells

The Search Path

- How does Unix find commands to execute?
  - If you specify a *full pathname*, the shell looks into that path for the executable
  - If you *specify just a filename*, the shell searches for it in the search path

```
obelix[12]% echo $PATH
/usr/local/java/bin:/usr/local/Tex/bin:/opt/SUNWspro/bin:/usr/ucb:/usr/bin:/
  usr/ccs/bin:/usr/local/bin:/usr/openwin/bin:/usr/sbin:/usr/sfw/bin:.
```

- The shell does not look for executables in your current directory *unless*
  - You specify it explicitly, e.g. `./a.out`
  - The **.** is specified in the path variable
- There may be multiple versions of the same command in your search path
- The shell searches in each directory of the `$PATH` in left to right order and executes the first version
- The command `which` locates a Unix command and display its pathname or alias (read `man which`)
- Be careful when you name your program "`test`"; ***why***?

Shell Startup

- When csh or tcsh is executed, it runs certain configuration files:
  - .login runs once when you log in
    - Contains one-time things like terminal setup
  - .cshrc runs before the execution of any [t]csh process
    - Sets lots of variables, e.g., PATH, MANPATH
- Other shells, such as sh or bash, use a different file (called .profile) to do similar things
- You can look at .login and .cshrc
- *Only modify the lines that you fully understand!*
- To reset your shell files, in case of an "accident", execute the command script: /usr/local/bin/reset.login.env

History (C-Style shells)

- In C-Style shells, history command displays the command history list with line numbers, e.g.,

```
obelix[32]% history
24  14:34   setenv my_environment "nice and clean"
25  14:34   echo $my_environment
26  14:34   set my_feeling = "I like Unix"
27  14:34   echo $my_feeling
28  14:35   echo $my_environment
29  14:35   echo $my_feeling
30  14:35   tcsh
31  14:35   echo $my_feeling
32  14:37   history
```

Note that: the commands that have been entered between tcsh and exit (*inclusive*) will not show up in the history output

History (C-Style shells)

- You can rerun a command line in the history
  - !! reruns last command
  - !str reruns the latest command beginning with str
  - !n (where n is a number) reruns command number n in the history list
- tcsh allows you to use arrow keys to wander the history list easily
- The length of the history list is determined by the variable history, likely set in your .cshrc file
- The variable savehist determines how much history to be saved in the file named in histfile for your next session; savehist and histfile variables are also likely set in your .cshrc file

```
set history=24
set savehist=10
set histfile=$home/.history.$HOSTTYPE
```

The alias Command (C-Style shells)

- alias command (to create/display aliases)
  - alias alias-name real-command
    - alias-name is one word
    - real-command can have spaces in it
- Any reference to alias-name invokes real-command
- Examples
  - alias rm rm -i
  - alias cp cp -i
  - alias mv mv -i
  - alias cls clear
  - alias ls /usr/bin/ls -CF
    - -C lists entries by column
    - -F shows the /, *, @ after file names using ls
- Your aliases can be put in your .cshrc file
- The command which can display the alias of a command, if any

The alias Command (C-Style shells)

- alias without any argument lists *all* existing aliases

```
obelix[33]% alias
bye        clear;logout
cls        clear
cp         (/usr/bin/cp -i)
h          history
laser      (lpr -Plw)
lo         logout
ls         (/usr/bin/ls -CF)
mv         (/usr/bin/mv -i)
print      lpr
rl         rlogin
tarlist    (tar tvf)
untar      (/usr/bin/tar xovf)
```

The unalias Command (C-Style shells)

- unalias command: delete aliase(s)
  - unalias alias-name
    - alias-name is one word
- unalias removes an alias-name from the alias list
- Examples
  - unalias rm
  - unalias cp
  - unalias mv
  - unalias cls
  - unalias ls

The unalias Command (C-Style shells)

- Example

```
obelix[34]% which ls
ls:     aliased to /usr/bin/ls -CF

obelix[35]% unalias ls
obelix[36]% which ls
/usr/ucb/ls

obelix[37]% alias ls ls -al
obelix[38]% which ls
ls:     aliased to ls -al

obelix[39]% unalias ls
obelix[40]% which ls
/usr/ucb/ls
```

Command and Filename Completion (C-Style shells)

- In tcsh and bash, you can let the shell complete a long command name by:
  - ☐ Typing a prefix of the command
  - ☐ Hitting the TAB key

  The shell will fill in the rest for you, if possible
- tcsh and bash can also complete file names:
  - ☐ Type first part of file name
  - ☐ Hit the TAB key

  The shell will complete the rest, if possible