COMP20200 Unix Programming

Lecture 20

Ravi Reddy Manumachu (ravi.manumachu@ucd.ie)

CS, University College Dublin, Ireland

19/04/2023

Overview

- UNIX process environment.
- Control flow using loops (while, until, for).
- Integer Arithmetic.

Process environment variables

Each process has an associated array of strings (name-value pairs), called its **environment**:

```
$ printenv
SHELL=/bin/bash
SESSION=ubuntu
USERNAME=manumachu
USER=manumachu
HOME=/home/manumachu
LANGUAGE=en_IE:en
...
```

Using environment variables in your scripts

```
# vars.sh
#!/bin/bash
echo "SESSION=$SESSION"
echo "USER=$USER"
echo "HOME=$HOME"
echo "LANGUAGE=$LANGUAGE"
exit 0
```

```
$ chmod u+x ./vars.sh && ./vars.sh
SESSION=ubuntu
USER=manumachu
HOME=/home/manumachu
LANGUAGE=en_IE:en
```

en_IE:en is the locale for English, Ireland.

Environment in process memory layout

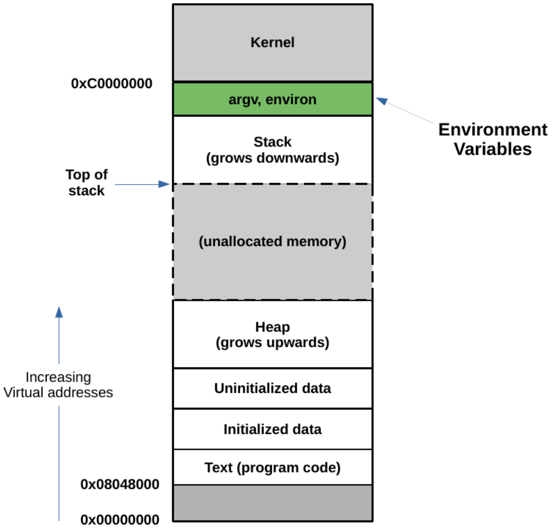

Figure: Environment variables in the memory layout of a process.

Where are the env variables set?

Each time you login to a UNIX system, shell scripts are run to set up default environment inherited by every process.

List of scripts that may exist:

```
/etc/profile        ~/.bash_profile
~/.bashrc           ~/.bash_login
~/.profile          ~/.bash_logout
```

To list hidden files (files beginning with "."), use **ls -al** command.

Script Sequence at Login

Following pseudo-code (not bash code) explains the sequence of scripts executed at login:

```
execute /etc/profile
IF ~/.bash_profile exists THEN
  execute ~/.bash_profile
ELSE
IF ~/.bash_login exists THEN
  execute ~/.bash_login
ELSE
IF ~/.profile exists THEN
  execute ~/.profile
END IF
END IF
END IF
```

First file found from the following list is executed:
~/.bash_profile, ~/.bash_login, and ~/.profile

~/.profile or $HOME/.profile

```
if [ -n "$BASH_VERSION" ]; then
  # include .bashrc if it exists
  if [ -f "$HOME/.bashrc" ]; then
    . "$HOME/.bashrc"
  fi
fi

# set PATH so it includes user's private bin if it exists
if [ -d "$HOME/bin" ] ; then
    PATH="$HOME/bin:$PATH"
fi
```

- `-n` True if string is not null.
- `-f` True if file exists and is a regular file.
- `-d` True if file exists and is a directory.

PATH Environment Variable

“Your path” is searched every time you execute a command.

- System wide default value set in `/etc/environment`

```
PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin
    :/sbin:/bin"
```

- Colon separated list, searched in order from left to right.
- First program found matching command is executed.
- Can override default by adding line to ~/.profile:

```
PATH="$HOME/bin:$PATH"
```

Shell prompt string

- The script \etc\profile sets the shell prompt string in PS1.
- Variable `PS1` is expanded and used as prompt string.

```
user@host:~/current_path$
```

- Also:
  - `PS2` secondary prompt string (for wrapping commands).
    Default: $>$
  - `PS3` prompt for the select command (interactive menus).
  - `PS4` For debugging scripts.
    Default: $+$

How to modify the environment?

Use “export” command to set environment variables.

```
$ export DOCHOME=/home/manumachu/documents

# vars2.sh
#!/bin/bash
echo "My documents folder=$DOCHOME"
exit 0

$ chmod u+x ./vars2.sh && ./vars2.sh
My documents folder=/home/manumachu/documents
```

Deleting environment variables

Use “unset” command to delete environment variables.

```
$ unset DOCHOME

# vars2.sh
#!/bin/bash
echo "My documents folder=$DOCHOME"
exit 0

$ chmod u+x ./vars2.sh && ./vars2.sh
My documents folder=
```

Lifetime of environment variables

- When you close a shell session, all the environment variables set in the session are lost.
- To make environment variables available for all your shell sessions, add them to your **.bashrc** script in your home directory.
- **.bashrc** script is executed when you login or when an interactive non-login shell is started.

```
$ ls -al $HOME/.bashrc
-rwxr-xr-x 1 ravi ravi 4050 Mar 31 17:14 /home/ravi/.
    bashrc
```

- When you start a shell session, all the exported variables in **.bashrc** script will be set in your shell environment.

Variables in your shell script

All your shell script variables (not the environment) are lost when your script terminates.

```
# vars3.sh
#!/bin/bash
DATABASE=mongodb
echo "My database=$DATABASE"
exit 0
```

```
$ chmod u+x ./vars3.sh && ./vars3.sh
My database=mongodb

$ echo "My database=$DATABASE"
My database=
```

Source command

Use **“source”** to execute your shell script.

```
#vars4.sh
#!/bin/bash
DATABASE=mongodb
echo "My database=$DATABASE"

$ source vars4.sh
My database=mongodb

$ echo "$DATABASE"
mongodb
```

Interaction between scripts

- Script "a.sh" sets a variable MSG.
- It then calls script "b.sh".

```
#a.sh
#!/bin/bash
MSG="hello"
./b.sh
```

- Script "b.sh" prints the contents of variable MSG.

```
#b.sh
#!/bin/bash
echo $MSG
```

Interaction between scripts

```
$ chmod u+x ./a.sh ./b.sh
$./a.sh
<no output>
```

- Problem is executing **"./b.sh"** launches a new shell to execute the script.
- MSG is not exported to the new shell environment.
- Two ways to fix this problem.

Solution 1: Using export

```
# First solution
#a.sh
#!/bin/bash
export MSG="hello"
./b.sh

$ ./a.sh
hello
```

Solution 2: Using source

```
# Second solution
#a.sh
#!/bin/bash
MSG="hello"
source b.sh

$ ./a.sh
hello
```

Control flow using Looping

While Loops

```
while condition; do list-of-commands; done
```

- Executes *list-of-commands* until *condition* no longer returns true.
- When *condition* fails, the script continues with the command following *done*.
- *Condition* can be any expression or command that returns a status.

Example:

```
#!/bin/bash
COUNTER=0
while [ $COUNTER -lt 10 ]; do
     echo The counter is $COUNTER
     let COUNTER=COUNTER+1
done
```

Until Loops

```
until condition; do list-of-commands; done
```

- Executes *list-of-commands* until *condition* returns true.
- Same as a while loop with an inverted condition.

```
while true; do sleep 1; done
```

- Will loop indefinitely.

```
until false; do sleep 1; done
```

- This will also loop indefinitely.

Fixed-Length For Loops

```
for item in <list>; do <commands>; done
```

- Expands *list* into a list of items.
- Iterates for each *item* and executes *commands*.

Example: Copy each file with *.txt* extension to file with *.txt.bak* extension.

```
for FILE in `ls *.txt`
do
   cp $FILE $FILE.bak
done
```

C-style Loops

Using bash C-style for loop.

```
#!/bin/bash
for (( COUNT=1; COUNT<=5; COUNT++ ))
do
    echo $COUNT
done

Output:

1
2
3
4
5
```

Integer Arithmetic

Integer Arithmetic

Partial list of operators available:

Syntax:	Meaning:
a++, $a--$	Post-increment/decrement (add/subtract 1)
++a, $--a$	Pre-increment/decrement
a+b, a-b	Addition/subtraction
a*b, a/b	Multiplication/division
a%b	Modulo (remainder after dividing)
a**b	Exponential

Shell Arithmetic: First builtin

Many ways to use arithmetic:

The "Let" Builtin

```
$ i=1; let VAR=i+5
$ echo $VAR
6
$ let VAR++
$ echo $VAR
7
```

- Allows arithmetic to be performed on shell variables.
- Each expression is evaluated according to the rules of shell arithmetic.

Shell Arithmetic: First builtin examples

```
$ a=4; b=2

$ let VAR1=a+b; let VAR2=a-b

$ echo $VAR1,$VAR2
6,2

$ let VAR3=a*b; let VAR4=a/b

$ echo $VAR3,$VAR4
8,2
```

Shell Arithmetic: modulo and power

```
$ let VAR5=a%b; let VAR6=a**b

$ echo $VAR5,$VAR6
0,16

$ let VAR7=++a; let VAR8=--a

$ echo $VAR7,$VAR8
5,4
```

Shell Arithmetic: Second builtin

Second way to use arithmetic:

Shell expansions

```
$[ EXPRESSION ]
```

- Calculates the result of EXPRESSION.
- Note the use of spaces.

```
$ echo $[ 10*9 ]
90
```

Shell Arithmetic: expr builtin

- **"expr"**

```
$ expr 4+2
4+2

#Note the use of spaces
$ expr 4 + 2
6
$ expr 4 \* 2
8
$ VAR=$( expr 10 - 3 )
$ echo $VAR
7
```

Shell Arithmetic: (( )) builtin

- Using double parentheses.
- Spaces inside parentheses are not necessary.

```
$ VAR=$(( 4 * 5 ))
$ echo $VAR
20
```

Recap

Topics covered today:

- Process environment and interaction of scripts with the environment variables.
- Control flow using loops (while, until, for).
- Integer Arithmetic.

Functions, arrays, wildcards, job control...

Q & A