Bash: Variables, customization, and string operations

Glenn Bruns
CSUMB

Lecture Objectives

After this lecture, you should be able to:

- use shell and environment variables
- customize your use of bash
- use basic bash string operations
- use commands df and du

Local and Environment variables

Environment variables (also known as “global variables”):

- inherited by child processes
- usually written in upper case
- important ones: PATH, HOME, TERM, PS1
- often setup at login

Shell variables (also known as “local variables”):

- are not inherited by a child shell
- concern “short-term working conditions”

Env. variables: viewing, using, setting

```
$ echo $HOME
/home/CLASSES/brunsglenn
$
$ printenv PATH
/usr/lib/qt-
3.3/bin:/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbi
n:/sbin:/home/CLASSES/brunsglenn/bin
$
$ ls $HOME
bash-cheat-sheet.txt  mlfq.csv          README-paging-policy
bin                   mlfq.py           README-scheduler
$
$ export TEMP=$HOME
$ printenv TEMP
/home/CLASSES/brunsglenn
$
```

use EXPORT to set new environment variable

EXPORT not needed to set existing environ. variable

Shell variables: viewing, setting, unsetting

```
$ echo $x

$ x=10
$ echo $x
10
$ unset x
$ echo $x
```

Command ‘set’ shows the values of all local AND global variables.

(‘set’ has many options – see the bash man page)

Comparing shell/environment vars

```
$ cat test.sh
#!/bin/bash
echo $x
$ ls -l test.sh
-rwxr-xr-x 1 brun1992 shell_faculty 20 Feb 14 14:44 test.sh
$ x=1
$ echo $x
$ 1
$ ./test.sh

$ export x
$ ./test.sh
1
```

Summary: variables

If you need a variable only within the current shell:

- use local variable
- example: `foo=baz`

If you need a variable in current shell and all descendants:

- use global variable
- example: `export FOO=baz`

If you want a variable that will be set in all bash sessions:

- define global variable in .bash_profile

Assigning command output

```
$ x=$(date)
$ echo $x
Tue Nov 3 13:53:59 PST 2015
$
$ x=`date`
$ echo $x
Tue Nov 3 13:55:43 PST 2015
$
```

$(command)

`command`   (old way)

Aliases

An alias provides a shortcut for a command

Examples:

```
# clear screen
alias c="clear"
# prevent accidental deletions
alias rm="rm -i"
# make executable
alias ax="chmod a+x"
```

I personally avoid aliases that change the behavior of existing commands

An alias is not a shell variable!

Alias only substituted when first word on command line.

Bash startup files

```
$ cd
$ ls -a | grep bash
.bash_history
.bash_logout
.bash_profile
.bashrc
```

~/.bash_profile – run once, at login. Use it to:

- source ~/.bashrc
- initialize environment variables
- do other stuff to be done only at login, like displaying long messages, or starting other programs

~/.bashrc – run when any shell is started. Use it to:

- define aliases and functions; initialize shell vars

Customizing bash

.bash_profile

```
source ~/.bashrc
# init environment vars
export PATH=$PATH:$HOME/bin
export PS1="$ "
```

.bashrc

```
alias c=clear
alias lsl='ls -l'
alias lsf='ls -f'
alias m=less
```

google ‘bash startup’ and you can get lots of other ideas on customizing your shell

Bash strings

```
$ x = awesome
-bash: x: command not found
$
$ x=awesome
$ echo $x
awesome
$
$ x="is this awesome?"
$ echo $x
is this awesome?
$
```

String length

```
$ x=awesome
$ echo $x
awesome
$
$ x
-bash: x: command not found
$
$ echo ${#x}
7
$
```

${#string}

Substrings

```
$ x=awesome
$ echo $x
awesome
$
$ echo ${x:4}
ome
$ echo ${x:3}
some
$
$ echo ${x:3:2}
so
$
```

${string:position}

${string:position:length}

Substitution

```
$ x="awesome.whatevs"
$ echo $x
awesome.whatevs
$
$ echo ${x/awe/}
some.whatevs
$
$ echo ${x/whatevs/txt}
awesome.txt

$ echo ${x//e/baz}
awbazsome.whatevs
```

${parameter/pattern/string}

replaces the first match only!

pattern is **not** a regular expression – it's a glob (as in 'file globbing')

Microquiz

```
$ # what is the result?
$ x = knurled
-bash: x: command not found
$
$ x=knurled
$ # what is the result?
$ echo {#x}
{#x}
$
$ # what is the result?
$ echo ${#x}
7
$
$ # what is the result?
$ echo ${x/k/}
nurled
$
```

Advanced: string removal by pattern

```
$ x=“awesome.whatevs”
$ echo $x
awesome.whatevs
$
$ echo ${x#aw}
esome.whatevs
$
$ echo ${x%.whatevs}
awesome
$
$ echo ${x%.whatevs}.txt
$ awesome.txt
$
$ x="/foo/fizzbuzz.bar"
$ y=${x%.bar}
$ echo ${y##*/}
fizzbuzz
```

`${string#pattern}` – remove shortest starting match

`${string%pattern}` – remove shortest ending match

`${string##pattern}` – remove longest starting match

Bash resources

- bash man page
- Advanced bash-scripting guide

www.tldp.org/LDP/abs

www.tldp.org/LDP/abs/html/string-manipulation.html

From the latter:

Bash supports a surprising number of string manipulation operations. Unfortunately, these tools lack a unified focus... This results in inconsistent command syntax and overlap of functionality, not to mention confusion.

Disk usage: commands 'df' and 'du'

df - report disk space usage

-h for “human readable” sizes, like 5.3G

```
$ df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/vda3        32G  6.4G   24G  22% /
/dev/vda1       578M  151M  398M  28% /boot
/dev/vdb1       262G  202M  249G   1% /home
/dev/vdc1        23G  2.2G   20G  10% /var
/dev/vdd1        42G   11G   29G  27% /home/CLASSES
```

du – estimate space usage

```
$ du -h
4.0K    ./public_html
44K     ./fall15/os/homework
60K     ./fall15/os
64K     ./fall15
188K    ./data1
66M     ./data
4.0K    ./Mail
24K     ./.emacs.d/auto-save-
list
28K     ./.emacs.d
16K     ./ctests/addresses
44K     ./ctests/ptrs
36K     ./ctests/assem
20K     ./ctests/gsh/backup
44K     ./ctests/gsh
188K    ./ctests/proc_api
344K    ./ctests
12K     ./mail
16K     ./bin
66M     .
```

This shows space used by current directory and all subdirectories

Summary

Local variables - scope is current shell

- x=1

Global variables - scope is current and descendant shells

- export x=1

Aliases are for command shortcuts

Use .bashrc and .bash_profile for bash customization

Commands introduced in this lecture:

- printenv, unset, export, alias, df, du