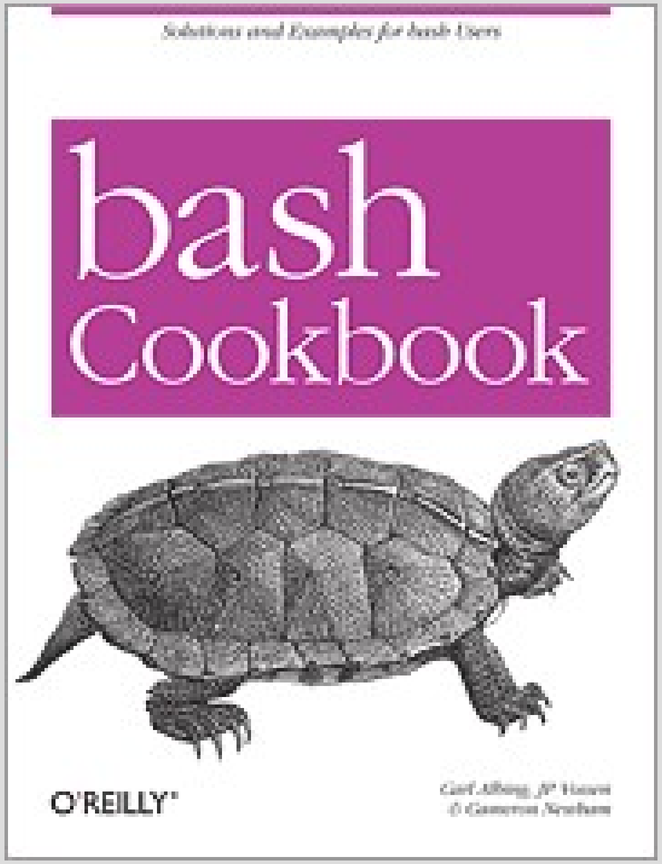
bash vs. dash

PLUG West
2008-10-20

PLUG North
2008-11-10

JP Vossen
bashcookbook.com


Solutions and Examples for bash Users
bash Cookbook
O'REILLY
Carl Albing, JP Vossen & Cameron Newham

STOLEN!!!

- Note: I stole a lot of this material from Carl Albing's "bash vs. dash" presentation at Ubuntu Line 2007!
- Original at: http://tinyurl.com/3mv8gy

bash vs. dash

- Huh?
- bash != Bourne != dash != ...
  http://en.wikipedia.org/wiki/Comparison_of_computer_shells
  http://en.wikipedia.org/wiki/Bash
  http://en.wikipedia.org/wiki/Bourne_shell
  http://en.wikipedia.org/wiki/Debian_Almquist_shell
- Why dash?
- The importance of */bin/sh*

bash vs. dash

- Syntax similarities
- Syntax differences
- Different uses
- Portability?

- /bin/sh --> dash, default user shell still bash
  - Ubuntu 6.10 or newer
    - https://wiki.ubuntu.com/DashAsBinSh
  - Debian Lenny or newer (proposed)
    - http://release.debian.org/lenny-goals.txt

bash vs dash

- bash
  - heavily interactive
  - feature rich
  - larger “footprint”
- dash
  - non-interactive
  - smaller “footprint”

Works in both

Grouping and subshells

```
echo $(ls)

( ls ; pwd ) | while read a b ; do  echo $a ; done

{ ls ; pwd ; } | while read a b ; do  echo $a ; done
```

Works in both

Arithmetic operator

- Must use *$var* in dash, but can omit the *$* in bash unless referring to a positional parameter (e.g., *$2*)

```
Y=$(($X+3))

Y=$(( $X + 3 ))
```

Works in both

Standard *for* loops

```
for i in 1 2 3 4 ; do echo $i ; done
```

```
for i in * ; do echo $i ; done
```

```
for i ; do echo $i ; done
```

Works in both

Standard *while* loops

```
while read a b ; do echo $a ; done
```

```
until read a b ; do echo $a ; done
```

Works in both

Standard *if/then/else* statements

```
if ls ; then pwd; else cd /tmp; fi

if ls
then
      pwd
elif cd /tmp
then
      echo ok
else
      echo no
fi
```

Works in both

Standard *case* statements

```
case $X in
  a) echo A ;;
  b) echo B ;;
  ?) echo other ;;
  *) echo default;;
esac
```

Works in both

Standard function definitions

Without *function* keyword !

```
# dash
foo ()
foo()

# bash
foo ()
foo()
function foo
function foo ()
```

Not available in dash

Conditional [[operator (shell glob on RHS)

  only the single [

Double == equality test

  only the single = allowed (POSIX)

```
# bash only
[[ $X == *.jpg ]] && echo "$X is a JPEG"
```

Not available in dash

- Numeric C-like *for* loop
  - But you can use *while* instead

```
for ((i=0; i<3; i++)); do ... ; done


i=0
while ($i < 3)
do
   ...
  ((i++))
done
```

Not available in dash

- dash avoids interactivity
  - tab completion!!!
  - history, edits!!!
  - menu builder select statement
  - 'help'

I/O redirection in dash

What works:

```
$ trash d.d
errmsg
$ trash d.d >/dev/null
errmsg
$ trash d.d 2>/dev/null
$ trash d.d >/dev/null 2>&1
$
```

I/O redirection in dash

What doesn't: redirecting both at once

```
# only in bash syntax:
$ trash d.d >&/dev/null
dash: Syntax error: Bad fd number
# dash interprets the '&' as a background cmd
$ trash d.d &>/dev/null
err
[1] + Done                    trash d.d
$
```

Close but not quite

Startup

bash:

uses $BASH_ENV when invoked (non-interactively)

```
BASH_ENV=$HOME/.alt_startrc
```

uses $ENV when invoked (interactively) as sh or in POSIX mode

dash:

uses $ENV

```
ENV=$HOME/.dashrc
```

Spot the problems?

```
#!/bin/bash -
# initialize databases from a standard file
# creating databases as needed.
DBLIST=$(mysql -e "SHOW DATABASES;" | tail +2)
select DB in $DBLIST "new..."
do
    if [[ $DB == "new..." ]]
    then
        printf "%b" "name for new db: "
        read DB rest
        echo creating new database $DB
        mysql -e "CREATE DATABASE IF NOT EXISTS $DB;"
    fi
    if [ "$DB" ]
    then
        echo Initializing database: $DB
        mysql $DB < ourInit.sql
    fi
    ((cnt++))
done
echo $cnt db initialized
```

Spot the problems?

```
#!/bin/bash -
# initialize databases from a standard file
# creating databases as needed.
DBLIST=$(mysql -e "SHOW DATABASES;" | tail +2)
select DB in $DBLIST "new..."
do
    if [[ $DB == "new..." ]]
    then
        printf "%b" "name for new db: "
        read DB rest
        echo creating new database $DB
        mysql -e "CREATE DATABASE IF NOT EXISTS $DB;"
    fi
    if [ "$DB" ]
    then
        echo Initializing database: $DB
        mysql $DB < ourInit.sql
    fi
    $((cnt++))
done
echo $cnt db initialized
```

checkbashisms?

- Now you tell us?!?
- *aptitude install devscripts*
- "Scripts to make the life of a Debian Package maintainer easier"
  - "checkbashisms: check whether a /bin/sh script contains any common bash-specific constructs"

checkbashisms!

```
$ checkbashisms bashisms.sh
possible bashism in bashisms.sh line 5 ('select' is not
   POSIX):
select DB in $DBLIST "new..."
possible bashism in bashisms.sh line 7 (alternative
   test command ([[ foo ]] should be [ foo ])):
    if [[ $DB == "new..." ]]
possible bashism in bashisms.sh line 7 (should be 'b
   = a'):
    if [[ $DB == "new..." ]]
possible bashism in bashisms.sh line 19 ('((' should
   be '$(('):
    ((cnt++))
```

Simple debugging works!

- *dash -n*
  - Like *bash -n* or *perl -c*, check basic syntax, but don't run
- *set -x*
  - debugging; show the final parsed command
- *set -v*
  - verbose; show the raw unparsed command

Summary

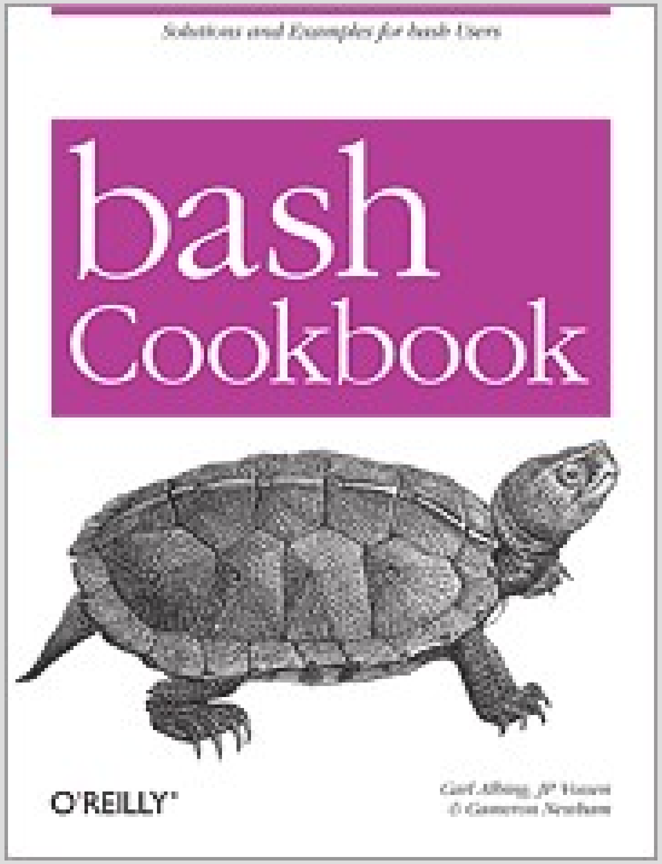
- bash and dash share a lot
  - basic function the same
  - basic syntax the same
  - simple debugging the same

- Dash excludes interactive features

- You can write portable scripts - carefully
  - avoid the exotic
  - do it in steps
  - when in doubt, try it out
  - use the *checkbashisms* script

Questions?

- Thanks to Carl for the original idea and material I stole...
- jp@jpsdomain.org
- I'm on the PLUG list
- http://bashcookbook.com/
- http://examples.oreilly.com/bashckbk/


Solutions and Examples for bash Users
bash
Cookbook
O'REILLY
Carl Albing, JP Vossen & Cameron Newham