GNU Bash

http://talk.jpnc.info/bash_scale11x.pdf

An Introduction to Advanced Usage

James Pannacciulli
Sysadmin @ (mt) Media Temple

Notes about the presentation:

This is a talk about Bash, not about GNU/Linux in general and not about the wealth of high quality command line utilities which are often executed from within Bash.

The assumed operating system is GNU/Linux, with a recent version of Bash. This talk is almost entirely Bash 3 compatible; I will try to point out any features or examples which require Bash 4.

I do not consider myself an *expert*. I am a **professional user** and an **enthusiast** and I want to share some of what I am learning, because Bash is a wonderful shell.

Command Types

File:

External executable file.

Builtin:

Command compiled in as part of Bash.

Keyword:

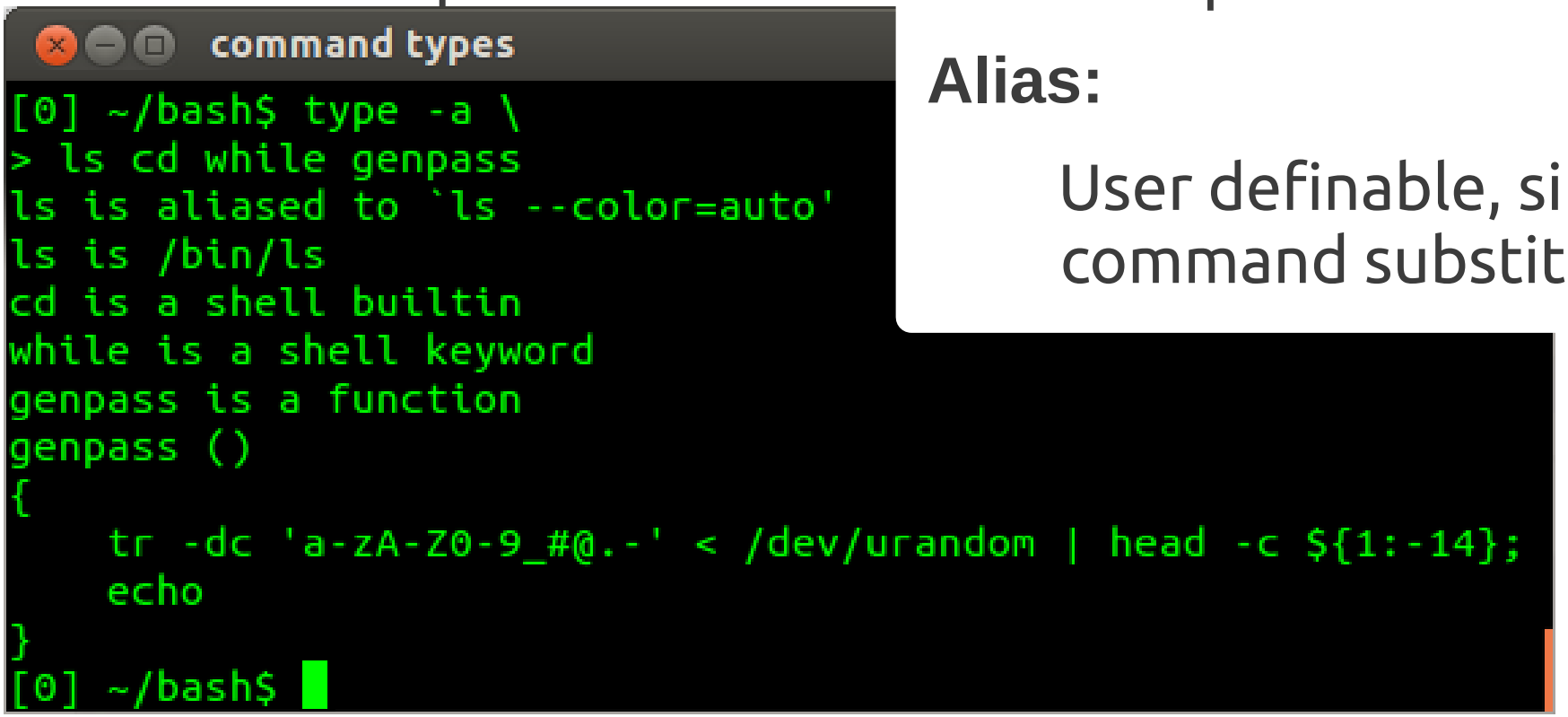
Reserved syntactic word.

Function:

User definable, named compound command.

Alias:

User definable, simple command substituion.



Getting Help

type:

Determine type of command, list contents of aliases and functions.

help:

Display usage information about Bash builtins and keywords.

apropos:

Search man pages.

man:

System manual.

info:

Advanced manual system primarily used for GNU programs.

General reference commands worth running:

man bash

man man

man -a intro

help

help help

info info

info

Some Useful Definitions

word Sequence of **characters** considered to be a single unit.

list Sequence of one or more **commands** or **pipelines**.

name A **word** consisting only of alphanumeric characters and underscores. Can not begin with a numeric character.

parameter An **entity** that stores **values**. A *variable* is a parameter denoted by a *name;* there are also *positional* and *special* parameters.

Compound Commands

Iteration:

Continuously loop over **list** of commands delineated by the keywords **do** and **done**.

while until for select

Conditionals:

Execute **list** of commands only if certain conditions are met.

if case

Command groups:

Grouped **list** of commands, sharing any external redirections and whose return value is that of the **list**.

(list) { list; }

While and Until Loops

while list1; do list2; done

Loop over **list2** of commands until **list1** returns a **non-zero** status.

until list1; do list2; done

Loop over **list2** of commands until **list1** returns a status of **0**.

The following construct is incredibly handy for processing lists of items: **while read**

For and Select Loops

for name in words; do list; done

Loop over **list** of commands, assigning **name** the value of each **word** until all **words** have been exhausted.

for ((expr1 ; expr2 ; expr3)); do list; done

Arithmetically Evaluate **expr1**, then loop over **list** of commands until **expr2** evaluates to **0**. During each iteration, evaluate **expr3**.

select name in words; do list; done

Create a menu item for each **word**. Each time the user makes a selection from the menu, **name** is assigned the value of the selected **word** and **REPLY** is assigned the **index** number of the selection.

Conditionals: if

if list1; then list2; fi

Evaluate **list1**, then evaluate **list2** only if **list1** returns a status of **0**.

if list1; then list2; else list3; fi

Evaluate **list1**, then evaluate **list2** only if **list1** returns a status of **0**. Otherwise, evaluate **list3**.

if list1; then list2; elif list3; then list4; else list5; fi

Evaluate **list1**, then evaluate **list2** only if **list1** returns a status of **0**. Otherwise, evaluate **list3**, then evaluate **list4** only if **list3** returns a status of **0**. Otherwise, evaluate **list5**.

Pattern Matching

Pattern matching is used in Bash for some types of **parameter expansion**, **pathname expansion**, *and the* **[[** *and* **case** *keywords.*

*****	Matches any string, including null.
?	Matches any single character.
[character class]	Matches any one of the characters enclosed between **[** and **]**.

The following predefined character classes are available with the **[:***class***:]** syntax:

alnum alpha ascii blank cntrl digit graph lower print punct space

Conditionals: case

```
case word in
    pattern1)
        list1;;
    pattern2 | pattern3)
        list2;;
esac
```

Match **word** against each **pattern** sequentially. When the first match is found, evaluate the **list** corresponding to that match and stop matching.

Command Groups

Subshell:

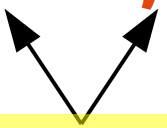
Evaluate **list** of commands in a subshell, meaning that its environment is distinct from the current shell and its parameters are contained.

(list)

Group command:

Evaluate **list** of commands in the current shell, sharing the current shell's environment.

{ list ; }



The **spaces** and **trailing semicolon** are *obligatory*.

Command and Process Substitution

Command substitution:

Replace the **command substitution** with the **output** of its **subshell**.

$(*list***)**

Process substitution:

Replace the **process substitution** with the location of a **named pipe** or **file descriptor** which is connected to the input or output of the **subshell**.

>(*list***)** **<(***list***)**

Parameters

Positional Parameters:

Parameters passed to command, encapsulating **words** on the command line as **arguments**.

$1 $2 $3 $4 $5 $6 $7 $8 $9 ${10} ${11} ...

Special Parameters:

Parameters providing **information** about positional parameters, the current shell, and the previous command.

$* $@ $# $- $$ $0 $! $? $_

Variables:

Parameters which may be **assigned values** by the user. There are also some special shell variables which may provide information, toggle shell options, or configure certain features.

name=*string*

For variable assignment, "=" must not have adjacent spaces.

Parameter Expansion: Conditionals

(check if variable is unset, empty, or non-empty)

	unset param	param=""	param="**gnu**"
${param-*default*}	default	–	gnu
${param=*default*}	*name=default*	–	gnu
${param+*alternate*}	–	alternate	alternate
${param?*error*}	error	–	gnu
Treat empty as unset:			
${param:-*default*}	default	default	gnu
${param:=*default*}	*name=default*	*name=default*	gnu
${param:+*alternate*}	–	–	alternate
${param:?*error*}	error	error	gnu

Parameter Expansion: Substrings

param="**racecar**"

Extraction: offset of **3**, length of **2**

${param:*offset*}	ecar
${param:*offset*:*length*}	ec
Removal from left edge:	pattern is '***c***'
${param#*pattern*}	ecar
${param##*pattern*}	ar
Removal from right edge:	pattern is '***c****'
${param%*pattern*}	race
${param%%*pattern*}	ra

Parameter Expansion:
Indirection, Listing, and Length

param="**parade**"; parade="**long**";
name=(**gnu not unix**); prefix is "**pa**"

Indirect expansion:

${!param} long

List names matching prefix:

${!prefix*} or "${!prefix@}" parade param

List keys in array:

${!name[*]} or "${!name[@]}" 0 1 2

Expand to length:

${#param} 6

Parameter Expansion: Pattern Substitution

param="**racecar**"

Substitution: pattern is '**c?**', string is '**T**'

${param/*pattern*/*string*} raTcar

${param//*pattern*/*string*} raTTr

Substitute at left edge: pattern is '**r**', string is '**T**'

${param/#*pattern*/*string*} Tacecar

Substitute at right edge:

${param/%*pattern*/*string*} racecaT

Tests

[expression] or test expression

Evaluate the expression with the **test** builtin command.

[[expression]]

Evaluate the expression with the **[[** keyword; word splitting and pathname expansion are **not** performed. Additionally, the righthand side of a string comparison (**==**, **!=**) is treated as a **pattern** when not quoted, and an additional regular expression operator, **=~**, is available.

-n string	string is **non-empty**
-z string	string is **empty**
string1 **==** string2	string1 and string2 are the **same**
string1 **!=** string2	string1 and string2 are **not the same**
-e file	file **exists**
-f file	file exists and is a **regular file**
-d file	file exists and is a **directory**
-t fd	fd is **open** and refers to a **terminal**

Arithmetic Expansion

((math and stuff))

name++ increment name after evaluation
name-- decrement name after evaluation

++name increment name before evaluation
--name decrement name before evaluation

- + * / % ** <= >= < > == != && ||

- Can be used as a test, returning 0 if comparison, equality, or inequality is true, or if the calculated number is not zero.
- Can provide in-line results when used like command substitution – **$((** *math* **))**.
- Bash does not natively support floating point.

Brace Expansion
Arbitrary String Generation

String generation:

prefix{*ab*,*cd*,*ef*}suffix

Sequence generation:

prefix{*x*..*y*}suffix

Sequencing by specified increment:

prefix{*x*..*y*..*incr*}suffix

Brace expansion may be **nested** and **combined**.

The **prefix** and **suffix** are optional.

Functions

Functions are compound commands which are defined in the current shell and given a function name, which can be called like other commands.

func.name () compound_cmd

Assign **compound_cmd** as function named **func.name**.

func.name () compound_cmd [>,<,>>] filename

Assign **compound_cmd** as function named **func.name**, which will always redirect to (**>**), from (**<**), or append to (**>>**) the specified filename.

Example code from the talk

```
while read var1 var2; do echo $var2 $var1; done
echo -e 'one two\none two three' > testfile
while read var1 var2; do echo $var2 $var1; done < testfile

for i in one two 'three four'; do echo "_-_-_-$i-_-_-_"; done

select choice in one two 'three four'; do echo "$REPLY : $choice"; done

if [ "a" == "a" ]; then echo "yep"; else echo "nope"; fi
if [ "a" == "b" ]; then echo "yep"; else echo "nope"; fi

case one in o) echo 'o';; o*) echo 'o*';; *) echo 'nope';; esac

unset x
(x=hello; echo $x); echo $x
{ x=hello; echo $x; }; echo $x

echo b; echo a | sort
(echo b; echo a) | sort
```

Example code from the talk

```
echo "$(echo "$(echo "$(echo "$(ps wwf -s $$)")")")"
echo this `echo quickly \`echo gets \\\`echo very \\\\\\\`echo ridiculous\\\\\\\`\\\`\``
echo "$(<testfile)"

PS1="[$?] $PS1"   # show exit status of prev. cmd in prompt
[ -t 0 ]
[ -t 2 ]
[ -t 2 ] 2>/dev/null
testvar="hello world"
[ $testvar == "hello world" ]  # fails
[ "$testvar" == "hello world" ]
[[ $testvar == "hello world" ]]
[[ $testvar == hello?w*d ]]

(( 0 ))
(( 1 ))
echo $(( 3 * 2 - (11 * 5) ))
```

Example code from the talk

```
echo bash{,e{d,s},ful{,ly,ness},ing}
echo {1..5}{0,5}%
echo {10..55..5}%
echo {a..z..12}
man{,}
cp -v filename{,.bak}   # quick backup of filename
```

Bash can actually complete (like tab completion) a list of files into nested brace expansion format with the **ESC-{** key combination. All key bindings may be displayed with the **bind -P** command.

Function examples

```
reverse ()
 for charlist
 do local arg
  while ((${#charlist}))
   do
    echo -n "${charlist:(-1)}"
    charlist="${charlist:0:(-1)}"
   done
 ((++arg == ${#@})) &&\
  echo ||\
  echo -n "${IFS:0:1}"
 done
```

Example usage:
reverse one two 'three four'

Function examples

```
memtop () {
for i in /proc/[0-9]*
  do
    echo -e "${i##*/}\t$(<$i/comm)\t$(pmap -d "${i##*/}" |\
      tail -1 | {
        read a b c mem d
        echo $mem
      }
    )"
  done |\
  sort -nr -k3 |\
  head -$((LINES - 3)) |\
  column -t
} 2>/dev/null
```

Example usage:
memtop
export -f memtop; watch bash -c memtop

A Few Good Links

- http://www.gnu.org/software/bash/
- http://tiswww.case.edu/php/chet/bash/NEWS
- http://tldp.org/LDP/abs/html/index.html
- http://wiki.bash-hackers.org/doku.php