Anatomy of the Command Line Part Deux: Examples Happen

PLUG
2026Feb12 @ 19:00
PLUG

der.hans (https://floss.social/@FLOX_advocate)
Database Support Engineering Manager
Rackspace DBaaS, nee Object Rocket
https://www.ObjectRocket.com/

ordering

1. redirection
2. pre-command variable assignment
3. expansion*
4. pipes
5. commands

ordering w/ expansion

- redirection
- pre-command variable assignment
- expansion
  1. brace expansion - { }
  2. tilde expansion - ~
  3. parameter and variable expansion - $VAR
  4. arithmetic expansion - $(( ))
  5. command substitution - $( )
  6. word splitting - IFS
  7. pathname expansion - globs
- pipes
- commands

presentation conventions

```
### use a semicolon to end a command and start another on the same line
;
# dollar sign indicates a user shell
$ I_am_me
### hash indicates a root shell
# I_have_the_power
### triple hash is for comments
```

```
$ command1
$ command2
$ command3
```

```
$ command1; command2; command3
```

```
# sync; sync; sync; reboot
```

redirection

```
$ echo "UseRoaming no" >> /etc/ssh/ssh_config
$ echo "UseRoaming no" | sudo tee -a /etc/ssh/ssh_config
```

```
$ echo anke >file.txt; grep anke file.txt
anke
$ echo anke >file.txt; grep anke file.txt | grep anke >file.txt
$
$ echo anke >file.txt; ls -l file.txt; grep anke file.txt | grep anke > file.txt; ls -l file.txt
-rw-r--r-- 1 lufthans lufthans 5  4. Jan 22:41 file.txt
-rw-r--r-- 1 lufthans lufthans 0  4. Jan 22:41 file.txt
$
```

pre-command variable assignments

```
$ LANG=C
$ find /ttmp
find: '/ttmp': No such file or directory
$ LANG=de_DE.UTF-8 find /ttmp | LANG=en_US sed -re 's/'
find: ‘/ttmp’: Datei oder Verzeichnis nicht gefunden
sed: -e expression #1, char 2: unterminated `s' command
$
```

expansion

- brace expansion
- TPCAP (done in a left-to-right fashion, then quote removal)
  - tilde expansion
  - parameter and variable expansion
  - command substitution
  - arithmetic expansion
  - process substitution (where supported, e.g. *NIX boxen)
- word splitting
- pathname expansion

tokenizing

From the bash man page:

> Only brace expansion, word splitting, and pathname expansion can increase the number of words of the expansion; other expansions expand a single word to a single word. The only exceptions to this are the expansions of "$@" and "${name[@]}", and, in most cases, $* and ${name[*]} as explained above (see PARAMETERS).

brace expansion

- bashism, not in sh
- left to right order is preserved

```
$ echo Number:{two,three,one}
Number:two Number:three Number:one
$ echo {,/usr}/{,s}bin
/bin /sbin /usr/bin /usr/sbin
$ echo {1..10}
1 2 3 4 5 6 7 8 9 10
$ ten=10; echo {1..$ten}
{1..10}
$ echo {1{0..9},20}
10 11 12 13 14 15 16 17 18 19 20
$ echo {a..e}
a b c d e
$ echo 1{0..9..2}
10 12 14 16 18
$
```

TPCAP

- expanded in the order encountered, left to right
  - tilde expansion
  - parameter and variable expansion
  - command expansion
  - arithmetic expansion
  - process substitution

tilde expansion

```
$ ls -ld ~/../../var/{lib,log,cache}
drwxr-xr-x 23 root root 4096 21. Jul 21:36 /home/plug/../../var/cache
drwxr-xr-x 71 root root 4096  9. Dez 02:16 /home/plug/../../var/lib
drwxr-xr-x 15 root root 4096  2. Jan 08:00 /home/plug/../../var/log
$ echo ~$USER
plug
$
```

- beware use in scripts

parameter expansion

- aka variables

```
$ echo $HOME
/home/plug
$ echo $horse_with_no_name
$
```

- lots of fun string manipulation that can be done during parameter expansion
  - search the bash manpage for :-

```
$ echo ${NOTHING:-something}
something
$ NOTHING=something
$ echo ${NOTHING:+lurch}
lurch
$ echo ${NOTHING^^}
SOMETHING
```

arithmetic expansion

- dollar sign, two opening parens, **integer expression**, two closing parens: **$((math))**
- normal math ordering
- the expression is implicitly double quoted

```
$ echo $(( 3 + 1 ))
4
```

- integer only

```
$ echo $(( 3 + 1.5 ))
bash: 3 + 1.5 : syntax error: invalid arithmetic operator (error token is ".5 ")
4
$ echo $(( 4 / 3 ))
1
$
```

math nesting

- nesting works, but use parenthesis for ordering

```
$ echo $(( $(( 3 + 5 )) * 2 ))
16
$ echo $(( ( 3 + 5 ) * 2 ))
16
```

- a double quote inside expression remains a double quote
- parameter and variable expansion, command substitution, and quote removal happen
- can leave sigil off variables

```
$ rush=2112; echo $(( 2 * $rush ))
4224
$ cent=19; year=84; orwell=$(( cent * 100 + year )); echo $orwell
1984
$
```

subshell

- opening paren, command list, closing paren: **(cmd1; cmd2)**
- runs inline
- creates a new env that doesn't change parent environment

```
$ somevar=someval; ( somevar=otherval ); echo $somevar
someval
$
```

- see Compound Commands in the bash manpage

command substitution

- dollar sign, opening paren, command list, closing paren: **$(cmd1; cmd2)**
- replaces the command with the results of the command
- creates a new env that doesn't change parent environment
- NOTE: do not use backticks
  - nesting with backticks is a quoting nightmare
- quotes are protected when using dollar paren

```
$ echo >file.$( date +%Y%b%d ).adoc
$ ls -l file.$( date +%Y%b%d ).adoc
-rw-r--r-- 1 lufthans lufthans 1 Sep 14 18:09 file.2023Sep14.adoc
$
```

```
cmdOut=$( command $( getArgs ) )
grepOut=$( grep -i PLUG $( find $HOME/ | grep -i plug | grep .adoc ) )
```

detours

- nesting: command substitution in arithmetic

```
$ echo $(( $( grep -c . /etc/motd ) * 2 ))
10
$
```

- nesting: command substitution with redirect shenanigans

```
$ echo -n $( echo foo >>file.txt ) >file.txt
$ ls -l file.txt
-rw-r--r-- 1 lufthans lufthans 0  9. Aug 23:10 file.txt
$ echo $( echo foo >>file.txt; ls -l file.txt ) >file.txt
$ cat file.txt
-rw-r--r-- 1 lufthans lufthans 4 9. Aug 23:11 file.txt
$
```

process substitution

- system must support named pipes (FIFOs) or /dev/fd
- use for commands that really, really want a filename rather than STDIN

```
$ diff <(ls /etc/ | tail -3) <(ls /etc/ | tail -4)
0a1
> NetworkManager
$
```

word splitting

- splits on text not in double quotes
- defaults to space, tab and newline
- IFS - Internal Field Separator
- scans results of parameter expansion, command substitution, and arithmetic expansion that wasn't quoted

```
$ ls /etc/{issue,motd}
/etc/issue  /etc/motd
$ echo ls /etc/{issue,motd}
ls /etc/issue /etc/motd
$
```

```
ls $( echo /etc/resolv.conf /etc/issue )
```

pathname expansion

- globbing

```
$ ls -d {/etc/w*,/usr/s*}
/etc/wgetrc      /etc/wpa_supplicant  /usr/share
/etc/wodim.conf  /usr/sbin            /usr/src
$ ls -d {/etc/w*,/usr/s*} | wc -w
6
$
```

pipes

- don’t use pipes for parallelization, they’re for redirecting STDOUT to STDIN

```
$ rm anke.txt; ls -l anke.txt >&2 | cat anke.txt >&2 | echo anke >anke.txt
anke
-rw-r--r-- 1 lufthans lufthans 5  6. Jan 23:33 anke.txt
$
```

```
$ ( LANG=en_US; find /ttmp 2>&1 | sed -re 's@ttmp@tmp@' 1>&2 | ls -l /etc/issue | date )
Thu Sep 14 17:30:56 MST 2023
find: '/tmp': No such file or directory
$
```

- see ${PIPESTATUS}

command

- and finally we run the command or equivalent

functions, aliases

- e.g. commands

extra credit

getting functional

```
$ cat /tmp/fx.sh
#!/bin/bash
function somefx() {
        somevar=otherval
        echo "fx: $somevar"
}
somevar=someval
fxout=$( somefx )
echo $fxout
echo $somevar
somefx
echo $somevar
$
$ /tmp/fx.sh
fx: otherval
someval
fx: otherval
otherval
$
```

functional one-liner

- let the squooshening begin
- note trailing semi-colons

```
$ ( function somefx() { somevar=otherval; echo "fx: $somevar"; }; somevar=someval; fxout=$(
somefx ); echo $fxout; echo $somevar; somefx; echo $somevar )
somevar )
fx: otherval
someval
fx: otherval
otherval
$
```

Does it blend?

```
$ cat ./resources/blend.sh
#!/bin/bash
function succeed() { return 0; }
function fail() { return 1; }
succeed
echo $?
fail
echo $?
fail
retVal=$?
if [ 0 -ne $retVal ] ; then
        echo "Hey look, 'fail' failed"
fi
$
$ ./resources/blend.sh
0
1
Hey look, 'fail' failed
$
```

grouping with curly brackets

- see { list; } under Compound Commands in the bash manpage
- runs in current environment, but groups the commands
- note trailing semi-colons

```
$ { echo fred; echo anke; } | sed -re 's/e/eee/'
freeed
ankeee
```

```
$ ( { grep y /usr/share/dict/american-english; grep z /usr/share/dict/ngerman; } | wc )
  63933   63933  856683
```

```
$ somevar=someval; { somevar=otherval; echo "curly: $somevar"; }; echo "nocurly: $somevar"
curly: otherval
nocurly: otherval
$
```

regex

- regexen are passed from the shell to the command
- **beware interpretation as globs**

```
$ touch f.{1,2,3}.txt
$ ls
f.1.txt  f.2.txt  f.3.txt
$ echo fff >file.txt
$ grep f.* file.txt
$ echo f.* file.txt
f.1.txt f.2.txt f.3.txt file.txt
$ grep f file.txt
fff
$
```

getting quoty

```
$ cowsay "There are 3 types of quotes"
 ______________________________
< There are 3 types of quotes >
 ------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

Tux says

```
$ cowsay -f tux "Also, there's cowsay"
 ______________________
< Also, there's cowsay >
 ----------------------
   \
    \
        .--.
       |o_o |
       |:_/ |
      //   \ \
     (|     | )
    /'\_   _/`\
    \___)=(___/
```

Go BIG

```
$ figlet -W "and figlet"
                    _      __  _            _          _
  __ _   _ __     __| |    / _| (_)   __ _  | |   ___  | |_
 / _` | | '_ \   / _` |   | |_  | |  / _` | | |  / _ \ | __|
| (_| | | | | | | (_| |   |  _| | | | (_| | | | |  __/ | |_
 \__,_| |_| |_|  \__,_|   |_|   |_|  \__, | |_|  \___|  \__|
                                      |___/
```

single quote before a command

```
$ \ls
```

```
$ alias ls="ls -r"
$ ls
file.txt  f.3.txt  f.2.txt  f.1.txt
$ \ls
f.1.txt  f.2.txt  f.3.txt  file.txt
$
```

```
$ echo this \
> is \
> one \
> line
this is one line
$
```

to apostrophe or to double apostrophe

```
$ echo "$SHELL"
/bin/bash
$ echo '$SHELL'
$SHELL
$
```

preserving returns

```
$ lines=$( grep bash /etc/passwd )
$ echo "$lines" | wc
      3       3     137
$ echo $lines | wc
      1       3     137
$
```

splitting

```
$ ( fy() { for i in $@; do echo $i; done; }; fz() { for i in "$@"; do echo $i; done; }; fy "one
two"; echo "<>"; fz "one two" )
one
two
<>
one two
```

space at beginning

```
$        echo

$
```

```
$ echo

$  ls
file.txt  f.3.txt  f.2.txt  f.1.txt
$ echo 2
2
$ history | tail -3
 2013  echo
 2014  echo 2
 2015  history | tail -4
$
```

QUIZ TIME !!!

examples

```
* redirection
* pre-command variable assignment
* expansion
** brace / tilde / parameter / arithmetic / cmd sub / process sub
** word splitting - IFS
** pathname expansion - globs
* pipes
* commands
```

- ls f*
- grep foo $(ls f* | grep -v w)
- LANG=en_US ls /t{tmp/*,etc/*,bin/*}
- ls -d ~/IMG*$((2112 + 1984))*{jpg,JPG,jpeg,JPEG}

examples zwei

```
* redirection
* pre-command variable assignment
* expansion
** brace / tilde / parameter / arithmetic / cmd sub / process sub
** word splitting - IFS
** pathname expansion - globs
* pipes
* commands
```

- ls -l file.out >file.out
- ls -l file.out | tee file.out
- sudo ls -l file.out >file.out

examples trois

```
* redirection
* pre-command variable assignment
* expansion
** brace / tilde / parameter / arithmetic / cmd sub / process sub
** word splitting - IFS
** pathname expansion - globs
* pipes
* commands
```

- (for i in $(find /etc -type f 2>/dev/null | grep sources.list); do tstamp=$(date -d@$(stat -c %Y $i) +%Y%b%d); echo cp -pi $i /tmp/$(echo "${i}" | tr / _).${tstamp}; done)

examples trois

```
* redirection
* pre-command variable assignment
* expansion
** brace / tilde / parameter / arithmetic / cmd sub / process sub
** word splitting - IFS
** pathname expansion - globs
* pipes
* commands
```

```
(
    for i in $( find /etc -type f 2>/dev/null | grep sources.list );
    do
        tstamp=$( date -d@$( stat -c %Y $i ) +%Y%b%d );
        echo cp -pi $i /tmp/$( echo "${i}" | tr / _ ).${tstamp};
    done
)
```

examples delta

```
* redirection
* pre-command variable assignment
* expansion
** brace / tilde / parameter / arithmetic / cmd sub / process sub
** word splitting - IFS
** pathname expansion - globs
* pipes
* commands
```

- echo $((2112 * 2)) $HOME | sed -re 's//z/g'
- $ exec ssh -t server export MYSQL_PS1="(\u@\${HOSTNAME%..}.\d)> "; export TERM=xterm; bash --rcfile <(echo export PS1=\"\\\u@\${HOSTNAME%..}\\\w\\\$ \") -i -o vi
- $ rsync "${@}" 2>&1 | (grep -Ev "^(file has vanished: |rsync warning: some files vanished before they could be transferred)" || true)

examples cinco

```
$ ps auxw | grep -F 'firefox -new-inst' | grep ^$USER | sed -re 's@pts/.*(firefox)@\1@'
lufthans   91354  0.0  0.0   9144  2192 firefox -new-inst
lufthans  531189  6.7  3.8 13913888 1210232 firefox -new-instance -P PLUG
lufthans 1445425  2.4  2.5 12585472 800120 firefox -new-instance -P Fediverse
```

```
$ ps auxw | grep -E '[f]irefox -new-inst' | grep ^$USER | sed -re 's@pts/.*(firefox)@\1@'
lufthans  531189  6.7  3.8 13914008 1206488 firefox -new-instance -P PLUG
lufthans 1445425  2.4  2.5 12585456 801116 firefox -new-instance -P Fediverse
$ ps auxw | grep -E 'firefox -+new-inst' | grep ^$USER | sed -re 's@pts/.*(firefox)@\1@'
lufthans  531189  6.7  3.8 13914008 1206488 firefox -new-instance -P PLUG
lufthans 1445425  2.4  2.5 12585456 801116 firefox -new-instance -P Fediverse
```

examples muttsu

I forget what 8 was for

```
$ ( r=2112; b=1984; echo $(( RANDOM / ( r + b ) )) )
6
$ ( r=2112; b=1984; echo $(( RANDOM / ( r + b ) )) )
5
$ ( r=2112; b=1984; echo $(( RANDOM / ( r + b ) )) )
1
```

examples ultimate

```
$ ( t="Anatomy of the Command Line"
> figlet -W "$(echo $t | awk '{print $3}' ; echo "   ${t##*n}${t:1:1}${t:$((${#t} - 6 )):1}" )"
)
  _      _
 | |_   | |__      ___
 | __|  | '_ \    / _ \
 | |_   | | | |  |  __/
  \__|  |_| |_|   \___|


                          _
          ___    _ __     __| |
         / _ \  | '_ \   / _` |
        |  __/  | | | | | (_| |
         \___|  |_| |_|  \__,_|
```

Resources

- bash man page
- Advanced Bash-Scripting Guide