Bash Cheat Sheet

Edit Cheat Sheet

See also Bash Regex Bash Associative Array Bash Functions

Syntax

File Operators

The complete list of bash 4.2 test operators:

```
       -a FILE        True if file exists.
       -b FILE        True if file is block special.
       -c FILE        True if file is character special.
       -d FILE        True if file is a directory.
       -e FILE        True if file exists.
       -f FILE        True if file exists and is a regular file.
       -g FILE        True if file is set-group-id.
       -h FILE        True if file is a symbolic link.
       -L FILE        True if file is a symbolic link.
       -k FILE        True if file has its `sticky' bit set.
       -p FILE        True if file is a named pipe.
       -r FILE        True if file is readable by you.
       -s FILE        True if file exists and is not empty.
       -S FILE        True if file is a socket.
       -t FD          True if FD is opened on a terminal.
       -u FILE        True if the file is set-user-id.
       -w FILE        True if the file is writable by you.
       -x FILE        True if the file is executable by you.
       -O FILE        True if the file is effectively owned by you.
       -G FILE        True if the file is effectively owned by your group.
       -N FILE        True if the file has been modified since it was last read.

       FILE1 -nt FILE2  True if file1 is newer than file2 (according to
                        modification date).

       FILE1 -ot FILE2  True if file1 is older than file2.

       FILE1 -ef FILE2  True if file1 is a hard link to file2.
```

String Operators

CHEAT SHEETS

Agile
Containers
DevOps Automation
DevOps Linux
DevOps Services
Languages
Network
Scripting
Security
Solutions
Virtualization
Windows
Glib Examples
Go Examples
Javascript Examples
PHP Examples

APIs
Bash Associative Array *
Bash Functions *
Bash Regex *
Bash
Color Distance
Shell Problems
Shell-Scripting
awk
packages.json
sed

The complete list of bash 4.2 string operators:

```
       -z STRING      True if string is empty.

       -n STRING
          STRING      True if string is not empty.

       STRING1 = STRING2
                      True if the strings are equal.
       STRING1 != STRING2
                      True if the strings are not equal.
       STRING1 < STRING2
                      True if STRING1 sorts before STRING2 lexicographically.
       STRING1 > STRING2
                      True if STRING1 sorts after STRING2 lexicographically.
```

String Manipulation

```
${str:position}         # substring starting at position
${str:position:len}     # substring starting at position with length len
${str#ubstring}         # delete shortest match from front
${str##substring}       # delete longest match from front
${str%substring}        # delete shortest match from back
${str%%substring}       # delete longest match from back
${str/pattern/replacement}  # pattern replace
${str/#pattern/replacement} # pattern replace at front
${str/%pattern/replacement} # pattern replace at end
${str//pattern/replacement} # global pattern replace
```

Arrays

Indexed arrays require no declaration

```
arr=("string 1", "string 2", "string 3")
arr=([1]="string 1", [2]="string 2", [3]="string 3")

arr[4]="string 4"
```

Check below under "Hashes" for accessing the different properties of an array.

Hashes

Since Bash v4

```
# Hashes need declaration!
declare -A arr

# Assigning values to associative arrays

arr[my key]="my value"
```

CHEAT SHEETS

Agile
Containers
DevOps Automation
DevOps Linux
DevOps Services
Languages
Network
Scripting
Security
Solutions
Virtualization
Windows
Glib Examples
Go Examples
Javascript Examples
PHP Examples

APIs
Bash Associative Array *
Bash Functions *
Bash Regex *
Bash
Color Distance
Shell Problems
Shell-Scripting
awk
packages.json
sed

```
arr["my key"]="my value"
arr[$my_key]="my value"

# Fetching values
echo ${arr[my key]}
echo ${arr["my key"]}
echo ${arr[$my_key]}

# Accessing the array
${arr[@]}         # Returns all indizes and their items (doesn't work with associative arrays)
${arr[*]}         # Returns all items
${!arr[*]}        # Returns all indizes
${#arr[*]}        # Number elements
${#arr[$n]}       # Length of $nth item

# Pushing to array
arr+=("new string value", "another new value")
```

Here Document

Bash allow here documents like this

```
cat <<EOT
[...]
EOT
```

To disable substitution in the here doc text quote the marker with single or double quotes.

```
cat <<'EOT'
```

To strip leading tabs use

```
cat <<-EOT
```

Debugging Scripts

For simple tracing add a

```
set -x
```

in the script or append the "-x" to the shebang or run the script like this

```
bash -x <script name>
```

As "set -x" enables tracing you can disable it with "set +x" again. This allows tracing only a part of the code (e.g. a condition in an inner loop). Additionally to "-x" you may want to set "-v" to see the shell commands that are executed. Combine both to

CHEAT SHEETS

Agile
Containers
DevOps Automation
DevOps Linux
DevOps Services
Languages
Network
Scripting
Security
Solutions
Virtualization
Windows
Glib Examples
Go Examples
Javascript Examples
PHP Examples

APIs
Bash Associative Array *
Bash Functions *
Bash Regex *
Bash
Color Distance
Shell Problems
Shell-Scripting
awk
packages.json
sed

```
set -xv
```

Writing Safer Scripts

Using

```
set -e
```

in a script you ensure that you never forget to check an exit code. Because if you do and the command calls returns an exit code != 0 the script just terminates. Of course you can also use it to not write checks if it is ok to just bail out.

Network Connections

```
# Establish a connection to 91.92.93.94:80 on file handle 4 with
if ! exec 4<> /dev/tcp/91.92.93.94/80; then
    echo "ERROR: Connection failed!"
fi

# Write something
echo -e "GET / HTTP/1.0\n" >&4

# Read something
cat <&4

# Close the socket
exec <&4-
exec >&4-
```

Simulate Reading From a File

Sometimes you might need to pass a file name when you want to pipe output from a commands. Then you could write to a file first and then used it, but you can also use the ">()" or "\<()" operator. This can be used with all tools that demand a file name paramter:

```
diff <(echo abc;echo def) <(echo abc;echo abc)
```

History

History Handling

Here are some improvements for the bash history handling:

```
unset HISTFILE      # Stop logging history in this bash instance
HISTIGNORE="[ ]*"   # Do not log commands with leading spaces
HISTIGNORE="&"      # Do not log a command multiple times

# Change up/down arrow key behaviour to navigate only similar commands
```

CHEAT SHEETS

Agile
Containers
DevOps Automation
DevOps Linux
DevOps Services
Languages
Network
Scripting
Security
Solutions
Virtualization
Windows
Glib Examples
Go Examples
Javascript Examples
PHP Examples

APIs
Bash Associative Array *
Bash Functions *
Bash Regex *
Bash
Color Distance
Shell Problems
Shell-Scripting
awk
packages.json
sed

```
bind '"\e[A":history-search-backward'
bind '"\e[B":history-search-forward'
```

Adding Timestamps

To add timestamps to your history set the following environment variable:

```
HISTTIMEFORMAT="%Y-%m-%d %H:%M:%S " # Log with timestamps
```

Easier History Navigation

If you do not like Ctrl-R to nagivate the history you can define other keys as PgUp and PgDown in /etc/inputrc:

```
    "\e[5~": history-search-backward
    "\e[6~": history-search-forward
```

History Hardening

For a secure bash configuration add the following settings to your global/users bashrc

```
HISTIGNORE=""
HISTCONTROL=""
HISTTIMEFORMAT='%Y-%m-%d %H:%M:%S '
HISTFILE=~/.bash_history
HISTFILESIZE=2000
readonly HISTFILE
readonly HISTSIZE
readonly HISTFILESIZE
readonly HISTIGNORE
readonly HISTCONTROL
readonly HISTTIMEFORMAT
shopt -s histappend
```

and finally mark the history file append only

```
chattr +a $HISTFILE
```

Misc

Command Completion

How to setup your own bash completion schemas. Here is a git example:

```
complete -W 'add branch checkout clone commit diff grep init log merge mv pull push rebase rm show status tag' git

complete -p    # To list defined completion schemes
```

CHEAT SHEETS

Agile
Containers
DevOps Automation
DevOps Linux
DevOps Services
Languages
Network
Scripting
Security
Solutions
Virtualization
Windows
Glib Examples
Go Examples
Javascript Examples
PHP Examples

APIs
Bash Associative Array *
Bash Functions *
Bash Regex *
Bash
Color Distance
Shell Problems
Shell-Scripting
awk
packages.json
sed

Note that the above example propably already comes prepared with your Linux distribution. You might want to check default definitions installed in /etc/bash_completion.d for a good starting point.

Kill all childs on exit

```
trap true TERM
kill -- -$$
```

Apply ulimit Changes Instantly

The problem behind this is documented in this blog post but it boils down to try to use the "-i" switch:

```
sudo -i -u <user>
```

If it doesn't work you might need to investigate and change the PAM configuration.

PS1: Escape Non-Print Chars

To avoid incorrect line break behaviour when editing the command line you need to escape control characters in PS1 like this:

```
\[color definition\]
```

For example:

```
\[\033[31m\] some text \[\033[0m\]
```

Comment on Disqus