Basic Admin Scripts

standby.cloud

https://standby.cloud/

Author: Georg Völl

Table of Contents

1 Management Summary 4

2 Installation and Uninstallation 5

2.1 Requirements for installation 5

2.2 Installation for all users 5

2.3 Installation for one user 5

2.4 Update and Uninstallation 6

2.5 Download and install or update only one script 6

2.6 Prevent update or deletion of a script after modification 6

3 General Use 7

3.1 Man Pages 7

3.2 Common Options 7

3.2.1 Option “--help” or “-h” 7
3.2.2 Option “--version” or “-v” 7
3.2.3 Option “--quiet” or “-q” 7
3.2.4 Option “--import” or “-i” 7
3.2.5 Option “--export” or “-e” 7
3.2.6 Option “--output” or “-o” 8

3.3 Common Environment Variables 8

3.3.1 Variable “LOGFILE” 8
3.3.2 Variable “ENVELOPE_TABLE” 8
3.3.3 Variable “DEBUG_LIB“ 9

3.4 Library with Variables, Functions and Aliases 9

3.5 Return Code 9

3.6 Handling of Devices and Signals 9

3.7 Output formats 9

3.7.1 Format “json” 10
3.7.2 Format “plain” 10
3.7.3 Format “keys” 11
3.7.4 Format “line” 11
3.7.5 Format “table” 11
3.7.6 Format “csv” 12
3.7.7 Format “tsv” 12
3.7.8 Format “etsv” 12

4 Description of the scripts 13

4.1 Script “errormsg” 13

4.2 Script “filecheck” 13

4.3 Script “confirm” 14

4.4 Script “transfer” 14

4.5 Script “check-sudo” 15

4.6 Script “check-port” 15

4.7 Script “get-ip” 15

4.8 Script “check-version” 16

4.9 Script “compare-version” 16

4.10 Script “convert-number” 17

4.11 Script “get-platform” 17
4.12 Script “print-header” 18
4.13 Script “print-table” 18
4.14 Script “select-table” 18
4.15 Script “norm-json” 19
4.16 Script “browse-json” 21
4.17 Script “convert-json” 21
5 Summary 22

Table of Figures

Figure 1: Displaying the help text 7
Figure 2: ENVELOPE_TABLE influences JSON output 8
Figure 3: JSON format 10
Figure 4: PLAIN format 10
Figure 5: KEYS format 11
Figure 6: LINE format 11
Figure 7: TABLE format 11
Figure 8: ETSV format 12
Figure 9: Script “errormsg” 13
Figure 10: Script “filecheck” 13
Figure 11: Script “confirm” 14
Figure 12: Script “transfer” 14
Figure 13: Script “check-sudo” 15
Figure 14: Script “check-port” 15
Figure 15: Script “check-version” 16
Figure 16: Script “compare-version” 16
Figure 17: Script “convert-number” 17
Figure 18: Script “get-platform” 17
Figure 19: Script “print-header” 18
Figure 20: Script “print-table” 18
Figure 21: Script “select-table” 19
Figure 22: Working with “norm-json” 20
Figure 23: Script “norm-json” 20
Figure 24: Script “browse-json” 21
Figure 25: Script “convert-json” 21

1 MANAGEMENT SUMMARY

This script collection is part of a larger collection designed to help administrators and developers complete upcoming tasks quickly and automatically. Great emphasis was placed on interoperability, which is why the scripts are written in BASH and run-on different platforms.

Why BASH and not Python or Java?

While software developers would probably prefer Python or Java, BASH know-how is still more widespread among administrators. Given this circumstance, the scripts were created in BASH so that administrators can understand the code and make changes if necessary.

Among other things, the scripts allow you to precisely determine the platform (to make your own scripts multi-platform capable), working with files is made easier, downloading from remote servers is improved, error output is simplified, and the processing of JSON files is optimized. Using them in your own scripts is easy and permitted without restrictions.

All scripts can be used free of charge and adapted if necessary. Updates and support are also free of charge. Send your questions, errors found and requests for enhancements to: support@standby.cloud

The latest version can be downloaded here:

https://standby.cloud/download/pdf/Basic-Admin-Scripts.pdf

2 INSTALLATION AND UNINSTALLATION

2.1 Requirements for installation

All scripts are written in BASH (a shell program and command language supported by the Free Software Foundation and first developed for the GNU Project) and should run on every UNIX / Linux environment (including macOS, Solaris and emulators like Windows Subsystem for Linux (WSL) or Cygwin).

All scripts are best tested on Oracle Linux 7 to 9 (which is compatible with RedHat), but also tested on macOS, Ubuntu, and Solaris. The processor architecture is irrelevant because we don't install binary tools. Older OS versions may not have all required tools or version of tools installed - so it is recommended to use a newer version of the OS. For example, we need “bash” equal or newer than version 4.3 and a newer version of "sed" for some string transformations. The scripts will inform you, when a required tool or version is missing (this will happen only in very few cases), and you should then update your tools or install GNU tools.

To download the scripts from the internet server, an internet connection and "curl" is required. In very few cases where "curl" is not installed, you can install or update "curl" manually. For example, on Oracle Linux (or Red Hat or CentOS) with the command:

```
sudo yum install curl
```

For the transformation of JSON files, the tool "jq" is needed. It is installed on most of Linux environments. If it is not already installed, please download, and install it from here:

https://jqlang.github.io/jq/download/

2.2 Installation for all users

To install all scripts (latest stable version) for all users (preferred), please call this command:

```
curl -skL https://standby.cloud/download/latest/install-scripts | sudo bash -s basic
```

You need to be logged in as “root” or enabled to use “sudo” – otherwise you can only install the scripts for one user (yourself). To update to newer versions, call the command from above again. Existing scripts were then being updated and obsolete scripts were deleted. After the installation, the scripts can be found at scripts path, the man pages at the man path and information about what was installed at info path.

```
SCRIPTSPATH: /usr/local/bin
    MANPATH: /usr/local/share/man
   INFOPATH: /usr/local/share/info
```

During the installation of the scripts, the profile “.bash_profile” will be copied to “.bash_profile.old” and the PATH will be extended if necessary. You will then be informed and asked to re-login or source the new profile. This is also the case if you install the scripts for a single user. Uninstallation of the scripts will not automatically change the profile back to its original.

2.3 Installation for one user

To install all scripts (latest stable version) for one user (yourself), please call this command:

```
curl -skL https://standby.cloud/download/latest/install-scripts | bash -s basic
```

This is necessary for example in OCI Cloud Shell, where “sudo” is not allowed. Used locations in this case are:

```
SCRIPTSPATH: ${HOME}/.local/bin
    MANPATH: ${HOME}/.local/share/man
   INFOPATH: ${HOME}/.local/share/info
```

Everything is installed under your HOME directory. To install the current beta version of the scripts, exchange “latest” with “beta” in the URL. This is only recommended if you know exactly what you are doing.

2.4 Update and Uninstallation

To update all the scripts, just use the same command as for installation. If you want to check first if there is a newer version available (or to see if the scripts are already installed or can be installed), use this command:

```
curl -skL https://standby.cloud/download/latest/install-scripts | sudo bash -s check
```

To uninstall all scripts and manuals, please call this command:

```
curl -skL https://standby.cloud/download/latest/install-scripts | sudo bash -s remove
```

To update or uninstall a single user installation, login with this user and call the above command without “sudo”. The information stored at the info path will be used to remove everything previously installed.

2.5 Download and install or update only one script

It is recommended to install or update the scripts with the above commands. All scripts are then installed or updated together, and dependencies are resolved. The latest stable version of the scripts can be found here:

https://standby.cloud/download/latest/

Older versions of the scripts can be found here:

https://standby.cloud/download/scripts/

If you only want to use a subset of the scripts or use an older version of one script, download it from the above URLs and install it manually. The links are also handy if you prefer to check first what you download before you install and use it. The full source code could be checked before installation.

2.6 Prevent update or deletion of a script after modification

If you want to modify one or more scripts by yourself (e.g., in case you find an error and you don’t want to wait for support to fix it or you want to enhance the functionality), it may make sense to save it then from unwanted deletion.

To do this, please look for a comment line in the script starting with “# Version: @(#)” and change it to something like “# Version: @(MyNameOrOrganization)”. When you do an update or uninstallation with the commands from above, your modified file will not be touched.

There is a risk that after an update some scripts will not work anymore because of the dependencies and shared interfaces that may change. So, it is highly recommended, that you tag your modifications and rename the modified files before the update. After the update, copy your modifications into the updated scripts. The tool “diff” could also help you to see the differences between your modified script and the new version of the script.

3 GENERAL USE

In this section you will find a description of the functionalities that are valid for all or most scripts.

3.1 Man Pages

All scripts are provided with a man page. Use the command "man" followed by the script name e.g.:

```
man get-ip
```

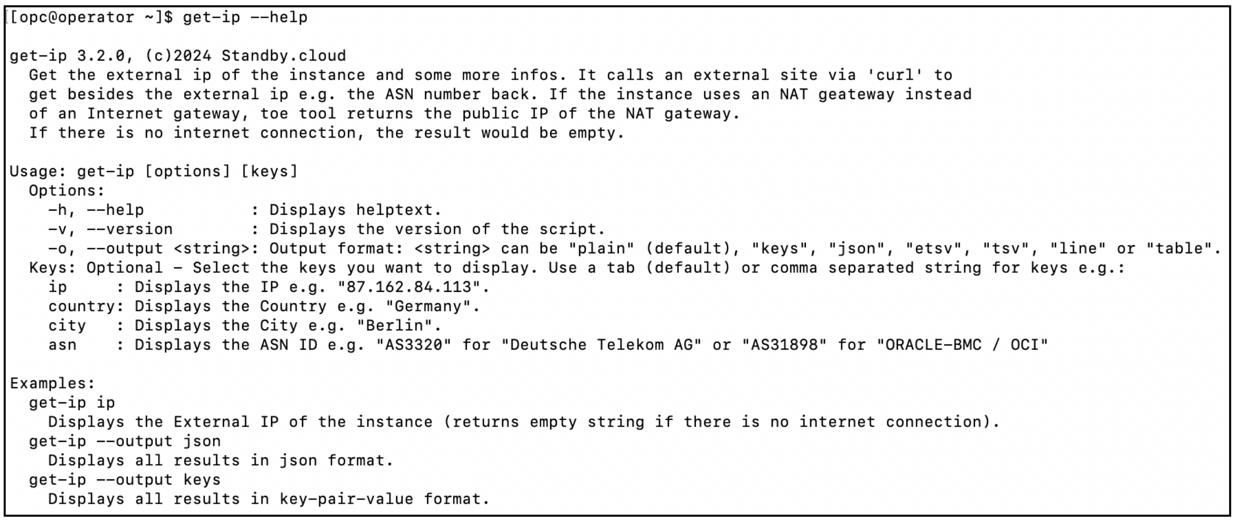
You will see the synopsis, description, options, examples and exit codes displayed for the script.

3.2 Common Options

In this section you will find an overview of the most used options.

3.2.1 Option "--help" or "-h"

If you call the script with this parameter, a short help will be displayed instead of executing the script.



```
[opc@operator ~]$ get-ip --help

get-ip 3.2.0, (c)2024 Standby.cloud
  Get the external ip of the instance and some more infos. It calls an external site via 'curl' to
  get besides the external ip e.g. the ASN number back. If the instance uses an NAT geateway instead
  of an Internet gateway, toe tool returns the public IP of the NAT gateway.
  If there is no internet connection, the result would be empty.

Usage: get-ip [options] [keys]
  Options:
    -h, --help           : Displays helptext.
    -v, --version        : Displays the version of the script.
    -o, --output <string>: Output format: <string> can be "plain" (default), "keys", "json", "etsv", "tsv", "line" or "table".
  Keys: Optional - Select the keys you want to display. Use a tab (default) or comma separated string for keys e.g.:
    ip     : Displays the IP e.g. "87.162.84.113".
    country: Displays the Country e.g. "Germany".
    city   : Displays the City e.g. "Berlin".
    asn    : Displays the ASN ID e.g. "AS3320" for "Deutsche Telekom AG" or "AS31898" for "ORACLE-BMC / OCI"

Examples:
  get-ip ip
    Displays the External IP of the instance (returns empty string if there is no internet connection).
  get-ip --output json
    Displays all results in json format.
  get-ip --output keys
    Displays all results in key-pair-value format.
```

Figure 1: Displaying the help text

3.2.2 Option "--version" or "-v"

If you call the script with this parameter, only the version of the script will be displayed e.g., "3.2.0".

3.2.3 Option "--quiet" or "-q"

Sometimes error messages are more annoying than helpful, e.g., if you only want to check the result of the entire script via return code and do not need detailed error messages. With this option, you can suppress most error messages via screen output.

3.2.4 Option "--import" or "-i"

Reads from a file whose name is specified after the option.

3.2.5 Option "--export" or "-e"

Directs the output to a file whose name is specified after the option.

3.2.6 Option “--output” or “-o”

Selects the output format which is specified after the option. Please lookup “Output formats” for more information. If you can specify the output format, you can usually also select the keys as a parameter to be displayed. These keys can be changed in order or additional keys can be specified as placeholders. Keys must be all in a string separated by colon (preferred), tab or comma e.g.:

```
get-ip --output json ip:user_agent/product
```

3.3 Common Environment Variables

The variables listed here influence the behavior of the scripts. Since all other scripts use "errormsg" and "lib.bash", and "print-table" is often used, these variables influence the overall behavior.

3.3.1 Variable “LOGFILE”

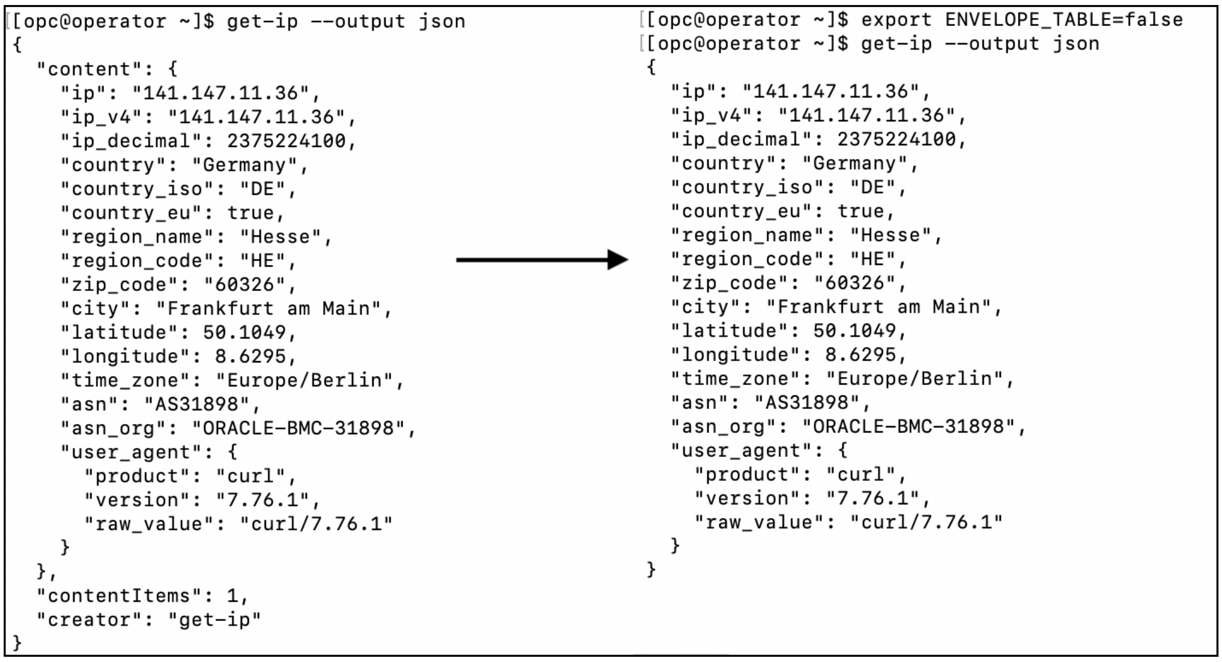
This environment variable is being used in the script “errormsg”. If set, all error messages, or warnings, displayed by the script, are also written to the specified file. This could be useful for debugging or for later analysis. Example:

```
export LOGFILE="${HOME}/test.log"
```

3.3.2 Variable “ENVELOPE_TABLE”

This environment variable is being used in the script “print-table”. If set to “false”, the output format “json” is not enveloped to “content” with number of items and optionally the name of the creator script. Example:

```
export ENVELOPE_TABLE=false
```



```
[opc@operator ~]$ get-ip --output json
{
  "content": {
    "ip": "141.147.11.36",
    "ip_v4": "141.147.11.36",
    "ip_decimal": 2375224100,
    "country": "Germany",
    "country_iso": "DE",
    "country_eu": true,
    "region_name": "Hesse",
    "region_code": "HE",
    "zip_code": "60326",
    "city": "Frankfurt am Main",
    "latitude": 50.1049,
    "longitude": 8.6295,
    "time_zone": "Europe/Berlin",
    "asn": "AS31898",
    "asn_org": "ORACLE-BMC-31898",
    "user_agent": {
      "product": "curl",
      "version": "7.76.1",
      "raw_value": "curl/7.76.1"
    }
  },
  "contentItems": 1,
  "creator": "get-ip"
}
```

```
[opc@operator ~]$ export ENVELOPE_TABLE=false
[opc@operator ~]$ get-ip --output json
{
  "ip": "141.147.11.36",
  "ip_v4": "141.147.11.36",
  "ip_decimal": 2375224100,
  "country": "Germany",
  "country_iso": "DE",
  "country_eu": true,
  "region_name": "Hesse",
  "region_code": "HE",
  "zip_code": "60326",
  "city": "Frankfurt am Main",
  "latitude": 50.1049,
  "longitude": 8.6295,
  "time_zone": "Europe/Berlin",
  "asn": "AS31898",
  "asn_org": "ORACLE-BMC-31898",
  "user_agent": {
    "product": "curl",
    "version": "7.76.1",
    "raw_value": "curl/7.76.1"
  }
}
```

Figure 2: ENVELOPE_TABLE influences JSON output

If JSON is enveloped with “content”, we are assuming that it was already normalized and not being transformed in combination with the script “norm_json”.

3.3.3 Variable “DEBUG_LIB“

This environment variable is being used in the library “lib.bash”. If set to “true” some variables are written to /dev/tty and optional written to $LOGFILE if variable is set. This could be useful for debugging or for later analysis. Example:

```
export DEBUG_LIB=true
```

3.4 Library with Variables, Functions and Aliases

All scripts use a common library "lib.bash". At the beginning of each script, the library is searched for and then integrated via "source". Useful functions, variables (e.g., color definitions) and alias definitions are defined. Although the library does not need to be executable (since it is included via "source"), do not change this because it will make it more difficult to find the library.

3.5 Return Code

Each script returns a return code indicating whether the script ran successfully or generated an error. You can also use this return code in your own scripts:

```
ip=`get-ip ip`
state=$?

if [ $state -eq 0 -a "$ip" != "" ]; then
       echo "Your external IP is: '$ip'"
else
       echo "IP not resolved (state: $state). Do you have an internet connection?"
fi
```

3.6 Handling of Devices and Signals

If the user presses Ctrl-C, the scripts were stopped immediately. This is done be setting “trap” for SIGINT in “lib.bash”. To avoid multiple abort messages in a possible cascading of scripts, a temporary file is created whose name begins with the username, e.g., /tmp/opc_INTERRUPT.tmp. This temp file is deleted as soon as a script is restarted. Also, most of the script are using one or more of these devices:

/dev/null	Null Device – Output is being ignored
/dev/tty	Terminal – No redirection (use the keyboard and screen)
/dev/stdin	Standard input – Could be a redirection e.g., through a pipe or from keyboard
/dev/stdout	Standard output – Could be the terminal or output to a file
/dev/stderr	Standard error – Could be the terminal or a logfile

If your platform does not support these devices, the scripts may not run properly.

3.7 Output formats

Some scripts can display the result in different output formats. You can find out which format is supported by which script in the description of the respective script itself. If a script itself does not allow output as "csv", for example, you can first select the "etsv" (Enhanced TSV) format and then use the "print-table" script to convert it to the desired final format.

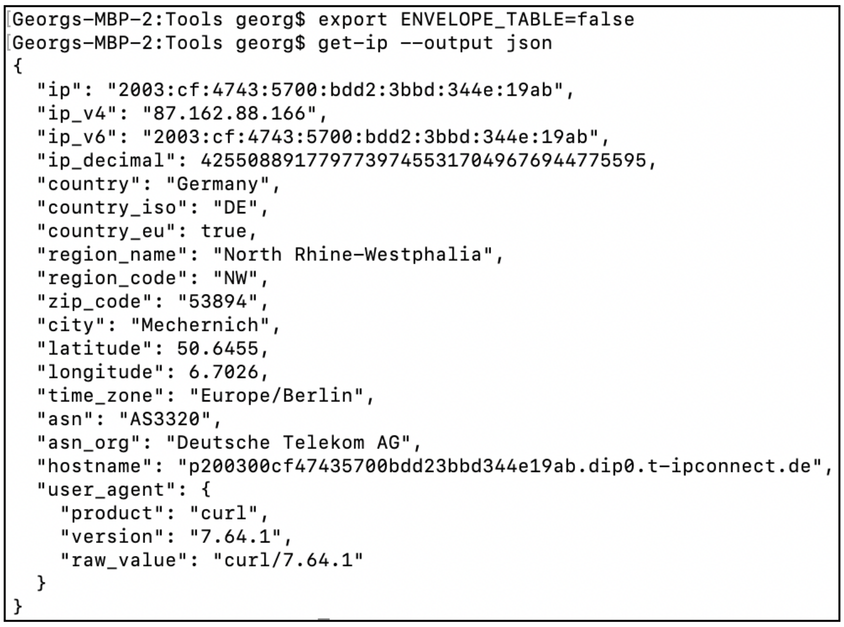
```
get-ip --output etsv | print-table --output csv
```

In addition, the values (if using format “line” or “table”) can be highlighted using colors. For example, if you want to display all values with content “true” in green and all values with content “false” in red, use this syntax:

```
get-ip --output etsv | print-table --output line --green true --red false
```

3.7.1 Format “json”

The output in JavaScript Object Notation (JSON) follows the rules of this notation. A sequence of key/value pairs is represented as an object or array according to a specific syntax. A value can consist of a value (string, number, or keyword such as null) or can be an object or array and thus represent another level with key/value pairs.

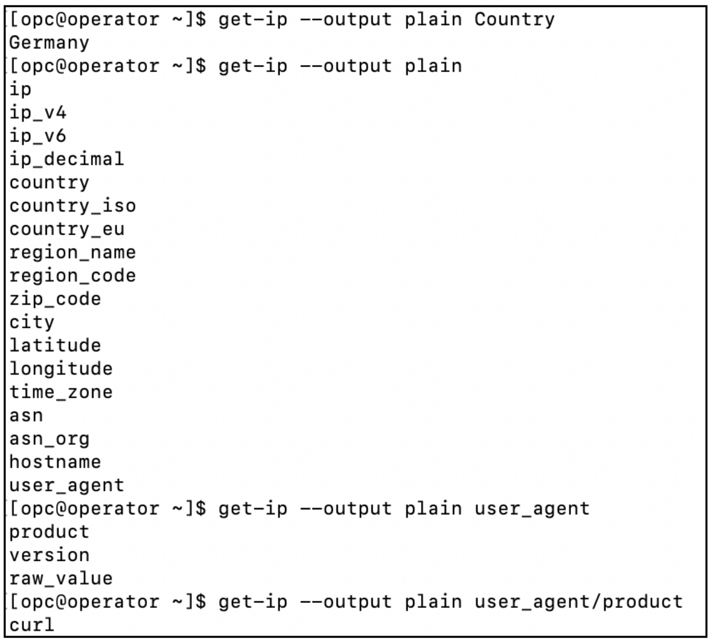


```
Georgs-MBP-2:Tools georg$ export ENVELOPE_TABLE=false
Georgs-MBP-2:Tools georg$ get-ip --output json
{
  "ip": "2003:cf:4743:5700:bdd2:3bbd:344e:19ab",
  "ip_v4": "87.162.88.166",
  "ip_v6": "2003:cf:4743:5700:bdd2:3bbd:344e:19ab",
  "ip_decimal": 42550889177977397455317049676944775595,
  "country": "Germany",
  "country_iso": "DE",
  "country_eu": true,
  "region_name": "North Rhine-Westphalia",
  "region_code": "NW",
  "zip_code": "53894",
  "city": "Mechernich",
  "latitude": 50.6455,
  "longitude": 6.7026,
  "time_zone": "Europe/Berlin",
  "asn": "AS3320",
  "asn_org": "Deutsche Telekom AG",
  "hostname": "p200300cf47435700bdd23bbd344e19ab.dip0.t-ipconnect.de",
  "user_agent": {
    "product": "curl",
    "version": "7.64.1",
    "raw_value": "curl/7.64.1"
  }
}
```

Figure 3: JSON format

3.7.2 Format “plain”

If you specify a single key in the "plain" format and this key has a value other than array or object, the value is displayed. This is the easiest way to get one value from one key. If there are several keys and none has been selected, then all keys of one level are displayed. If you want to display the value of a key on a lower level, first enter the key of the first level and then the key of the second level, separated by a slash.

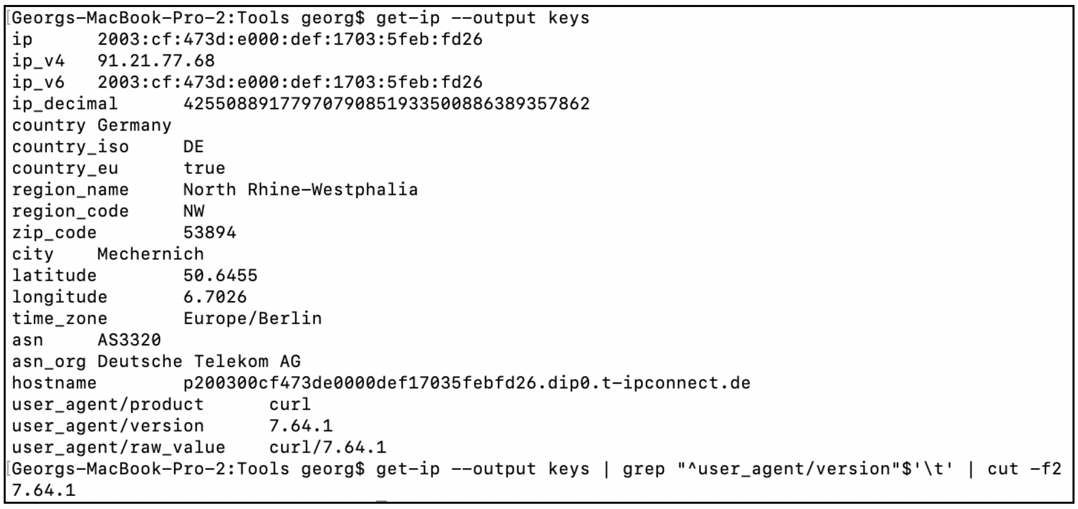


```
[opc@operator ~]$ get-ip --output plain Country
Germany
[opc@operator ~]$ get-ip --output plain
ip
ip_v4
ip_v6
ip_decimal
country
country_iso
country_eu
region_name
region_code
zip_code
city
latitude
longitude
time_zone
asn
asn_org
hostname
user_agent
[opc@operator ~]$ get-ip --output plain user_agent
product
version
raw_value
[opc@operator ~]$ get-ip --output plain user_agent/product
curl
```

Figure 4: PLAIN format

3.7.3 Format “keys”

The "keys" format is best suited to display all key/value pairs simultaneously and to use them for automated evaluation. A key can be searched for using "grep" and the value can be read using "cut" after the tab.

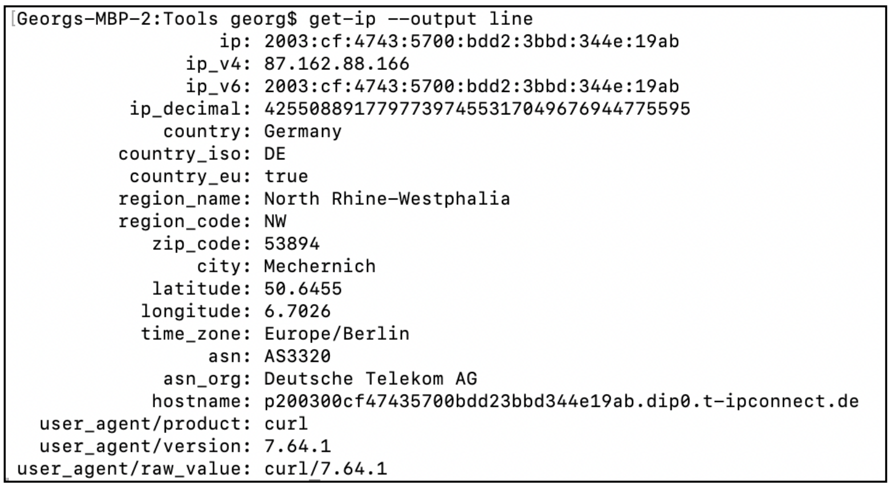


```
Georgs-MacBook-Pro-2:Tools georg$ get-ip --output keys
ip      2003:cf:473d:e000:def:1703:5feb:fd26
ip_v4   91.21.77.68
ip_v6   2003:cf:473d:e000:def:1703:5feb:fd26
ip_decimal      42550889177970790851933500886389357862
country Germany
country_iso     DE
country_eu      true
region_name     North Rhine-Westphalia
region_code     NW
zip_code        53894
city    Mechernich
latitude        50.6455
longitude       6.7026
time_zone       Europe/Berlin
asn     AS3320
asn_org Deutsche Telekom AG
hostname        p200300cf473de0000def17035febfd26.dip0.t-ipconnect.de
user_agent/product      curl
user_agent/version      7.64.1
user_agent/raw_value    curl/7.64.1
Georgs-MacBook-Pro-2:Tools georg$ get-ip --output keys | grep "^user_agent/version"$'\t' | cut -f2
7.64.1
```

Figure 5: KEYS format

3.7.4 Format “line”

Like "keys", the "line" format displays all key/value pairs simultaneously. However, automatic evaluation is more difficult because the focus here is on the output on the screen in a formatted way. Also (unlike "keys") only the key/value pairs that are set are displayed. For example, “ip_v6” and “hostname” are only displayed when returned from “get_ip”.

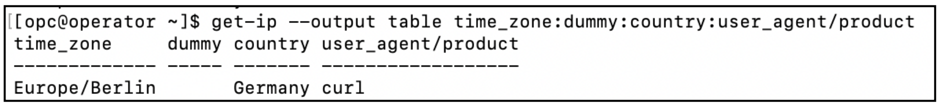


```
Georgs-MBP-2:Tools georg$ get-ip --output line
                  ip: 2003:cf:4743:5700:bdd2:3bbd:344e:19ab
               ip_v4: 87.162.88.166
               ip_v6: 2003:cf:4743:5700:bdd2:3bbd:344e:19ab
          ip_decimal: 42550889177977397455317049676944775595
             country: Germany
         country_iso: DE
          country_eu: true
         region_name: North Rhine-Westphalia
         region_code: NW
            zip_code: 53894
                city: Mechernich
            latitude: 50.6455
           longitude: 6.7026
           time_zone: Europe/Berlin
                 asn: AS3320
             asn_org: Deutsche Telekom AG
            hostname: p200300cf47435700bdd23bbd344e19ab.dip0.t-ipconnect.de
  user_agent/product: curl
  user_agent/version: 7.64.1
user_agent/raw_value: curl/7.64.1
```

Figure 6: LINE format

3.7.5 Format “table”

The "table" format is only intended for output on the screen in a formatted way. The keys form the heading, and the values are displayed below in tabular form.



```
[opc@operator ~]$ get-ip --output table time_zone:dummy:country:user_agent/product
time_zone     dummy country user_agent/product
------------- ----- ------- ------------------
Europe/Berlin       Germany curl
```

Figure 7: TABLE format

3.7.6 Format “csv”

CSV (comma-separated values) is a text file format that uses commas to separate values, and newlines to separate records. A CSV file stores tabular data (numbers and text) in plain text, where each line of the file typically represents one data record. Each record consists of the same number of fields, and these are separated by commas in the CSV file. If the field delimiter itself may appear within a field, fields can be surrounded with quotation marks.

3.7.7 Format “tsv”

TSV (tab-separated values) is a simple, text-based file format for storing tabular data. Records are separated by newlines, and values within a record are separated by tab characters. The TSV format is thus a delimiter-separated values format, like comma-separated values.

TSV is a simple file format that is widely supported, so it is often used in data exchange to move tabular data between different computer programs that support the format. For example, a TSV file might be used to transfer information from a database to a spreadsheet.

The first row (header) contains the column headings. All subsequent rows contain the actual data.

3.7.8 Format “etsv”

ETSV (enhanced tab-separated values) is based on TSV and is compatible with it, but at the same time allows more information to be stored about the individual value. This makes it possible to create a JSON file from an ETSV.

In a JSON file, the key-value pair is enclosed in quotation marks if the value is a string e.g.:

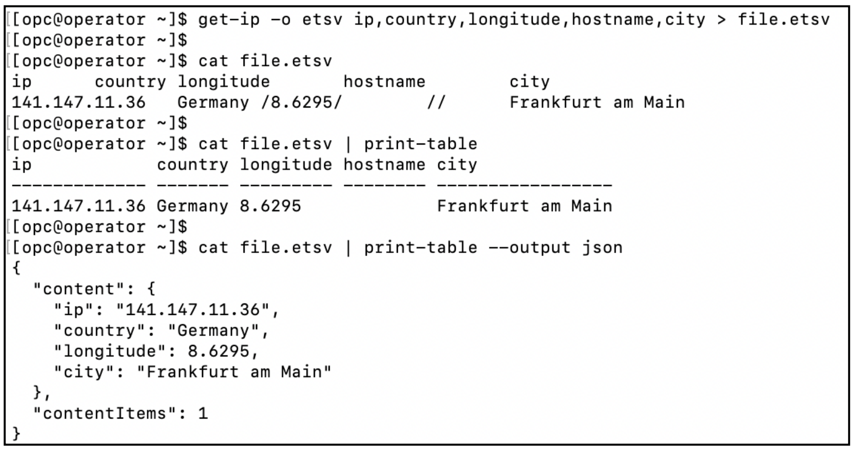
```
"country": "Germany"
```

However, if the value is a number, it is not enclosed in quotation marks e.g.:

```
"longitude": 6.7026
```

In ETSV, a number is enclosed with a slash so that this can be used when converting to JSON and the quotation marks are omitted in this case. For example:

```
abc    will be converted to “abc”
/123/  will be converted to 123
/{}/   (JSON Object) will be converted to {}
/[]/   (JSON Array) will be converted to []
/null/ will be converted to null
//     will be skipped (key/value pair will be ignored during conversion)
```



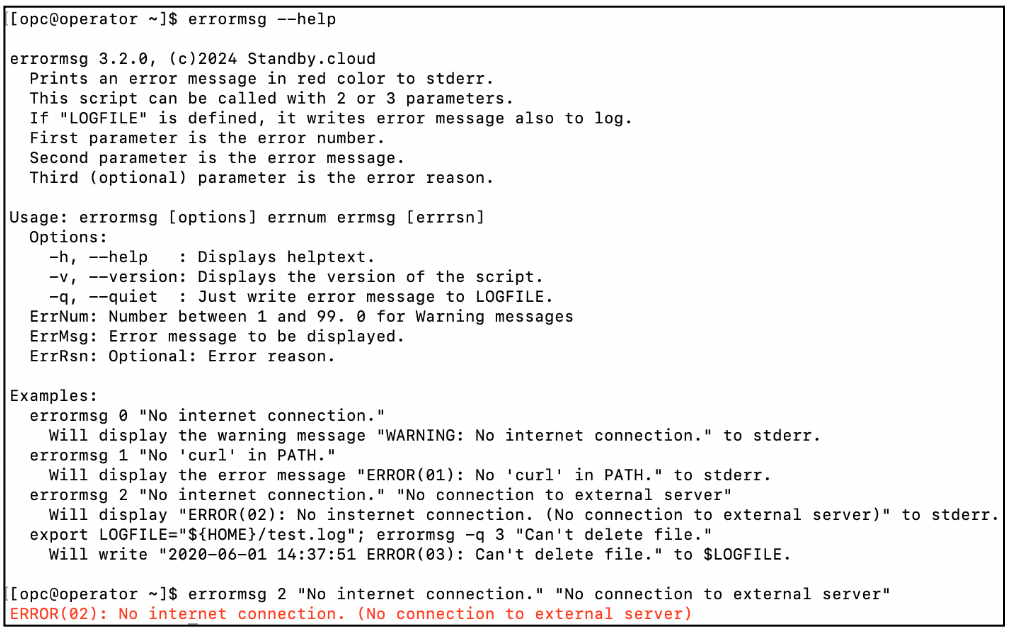
```
[opc@operator ~]$ get-ip -o etsv ip,country,longitude,hostname,city > file.etsv
[opc@operator ~]$
[opc@operator ~]$ cat file.etsv
ip      country longitude       hostname        city
141.147.11.36   Germany /8.6295/        //      Frankfurt am Main
[opc@operator ~]$
[opc@operator ~]$ cat file.etsv | print-table
ip            country longitude hostname city
------------- ------- --------- -------- -----------------
141.147.11.36 Germany 8.6295             Frankfurt am Main
[opc@operator ~]$
[opc@operator ~]$ cat file.etsv | print-table --output json
{
  "content": {
    "ip": "141.147.11.36",
    "country": "Germany",
    "longitude": 8.6295,
    "city": "Frankfurt am Main"
  },
  "contentItems": 1
}
```

Figure 8: ETSV format

4 DESCRIPTION OF THE SCRIPTS

4.1 Script “errormsg”

This script helps to output error messages and is used in all other scripts.

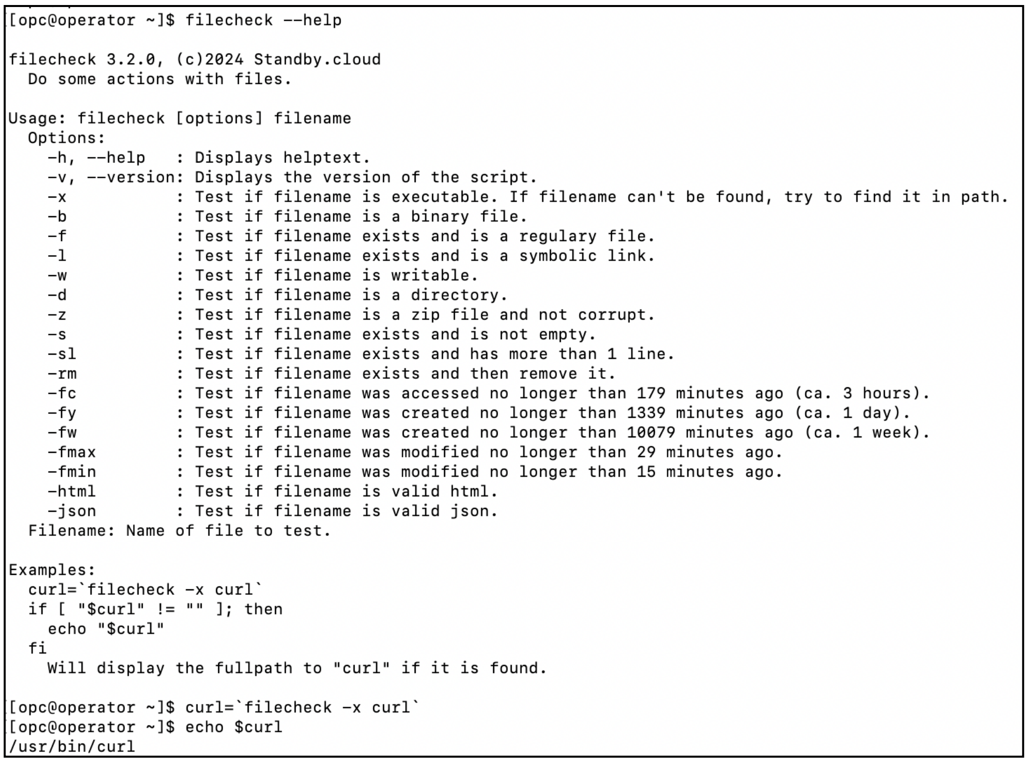


```
[opc@operator ~]$ errormsg --help

errormsg 3.2.0, (c)2024 Standby.cloud
  Prints an error message in red color to stderr.
  This script can be called with 2 or 3 parameters.
  If "LOGFILE" is defined, it writes error message also to log.
  First parameter is the error number.
  Second parameter is the error message.
  Third (optional) parameter is the error reason.

Usage: errormsg [options] errnum errmsg [errrsn]
  Options:
    -h, --help   : Displays helptext.
    -v, --version: Displays the version of the script.
    -q, --quiet  : Just write error message to LOGFILE.
  ErrNum: Number between 1 and 99. 0 for Warning messages
  ErrMsg: Error message to be displayed.
  ErrRsn: Optional: Error reason.

Examples:
  errormsg 0 "No internet connection."
    Will display the warning message "WARNING: No internet connection." to stderr.
  errormsg 1 "No 'curl' in PATH."
    Will display the error message "ERROR(01): No 'curl' in PATH." to stderr.
  errormsg 2 "No internet connection." "No connection to external server"
    Will display "ERROR(02): No insternet connection. (No connection to external server)" to stderr.
  export LOGFILE="${HOME}/test.log"; errormsg -q 3 "Can't delete file."
    Will write "2020-06-01 14:37:51 ERROR(03): Can't delete file." to $LOGFILE.

[opc@operator ~]$ errormsg 2 "No internet connection." "No connection to external server"
ERROR(02): No internet connection. (No connection to external server)
```

Figure 9: Script “errormsg”

4.2 Script “filecheck”

This script helps with working with files (uses and extends the existing possibilities in BASH).



```
[opc@operator ~]$ filecheck --help

filecheck 3.2.0, (c)2024 Standby.cloud
  Do some actions with files.

Usage: filecheck [options] filename
  Options:
    -h, --help   : Displays helptext.
    -v, --version: Displays the version of the script.
    -x           : Test if filename is executable. If filename can't be found, try to find it in path.
    -b           : Test if filename is a binary file.
    -f           : Test if filename exists and is a regulary file.
    -l           : Test if filename exists and is a symbolic link.
    -w           : Test if filename is writable.
    -d           : Test if filename is a directory.
    -z           : Test if filename is a zip file and not corrupt.
    -s           : Test if filename exists and is not empty.
    -sl          : Test if filename exists and has more than 1 line.
    -rm          : Test if filename exists and then remove it.
    -fc          : Test if filename was accessed no longer than 179 minutes ago (ca. 3 hours).
    -fy          : Test if filename was created no longer than 1339 minutes ago (ca. 1 day).
    -fw          : Test if filename was created no longer than 10079 minutes ago (ca. 1 week).
    -fmax        : Test if filename was modified no longer than 29 minutes ago.
    -fmin        : Test if filename was modified no longer than 15 minutes ago.
    -html        : Test if filename is valid html.
    -json        : Test if filename is valid json.
  Filename: Name of file to test.

Examples:
  curl=`filecheck -x curl`
  if [ "$curl" != "" ]; then
    echo "$curl"
  fi
    Will display the fullpath to "curl" if it is found.

[opc@operator ~]$ curl=`filecheck -x curl`
[opc@operator ~]$ echo $curl
/usr/bin/curl
```

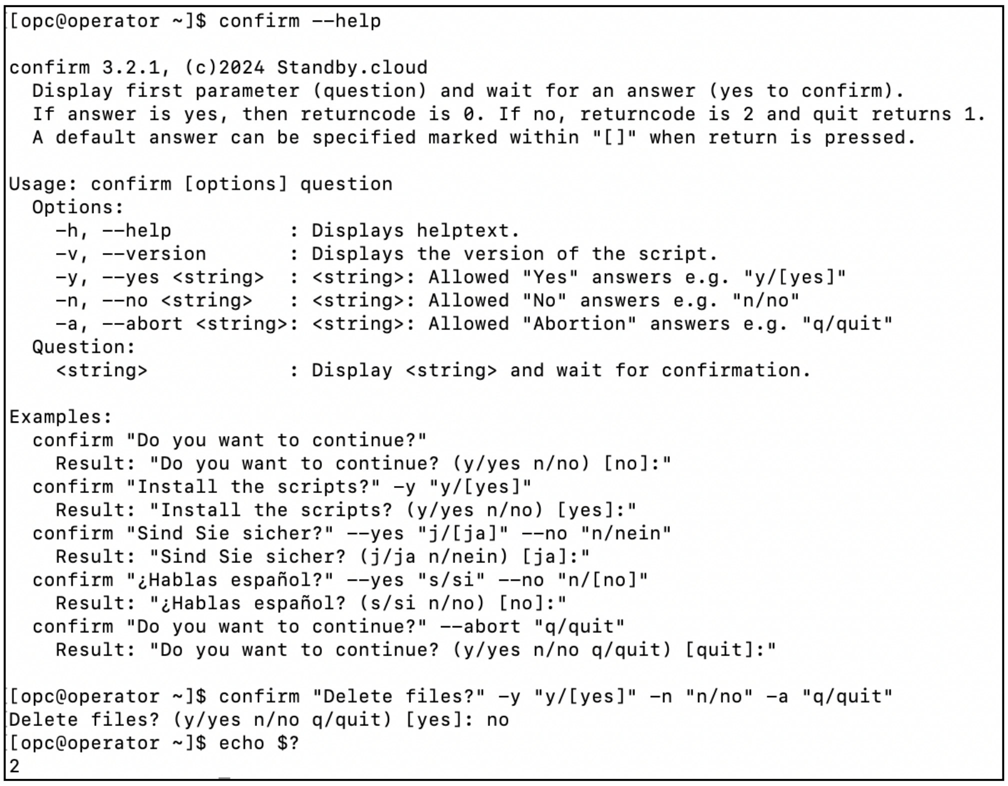
Figure 10: Script “filecheck”

4.3 Script “confirm”

With this script, you can ask a multiple-choice question and get the answer via /dev/stdin. The input can therefore be made via the keyboard or, for example, via "pipe":

```
echo "yes" | confirm "Do you want to continue?"
```

Returns the value "0" as if the input had been made via the keyboard.

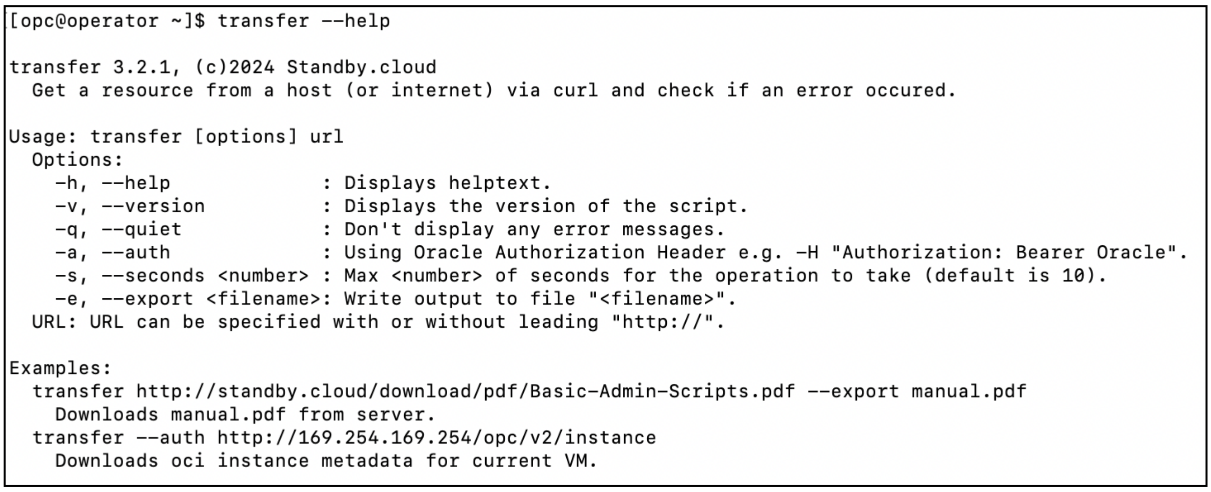


```
[opc@operator ~]$ confirm --help

confirm 3.2.1, (c)2024 Standby.cloud
  Display first parameter (question) and wait for an answer (yes to confirm).
  If answer is yes, then returncode is 0. If no, returncode is 2 and quit returns 1.
  A default answer can be specified marked within "[]" when return is pressed.

Usage: confirm [options] question
  Options:
    -h, --help          : Displays helptext.
    -v, --version       : Displays the version of the script.
    -y, --yes <string>  : <string>: Allowed "Yes" answers e.g. "y/[yes]"
    -n, --no <string>   : <string>: Allowed "No" answers e.g. "n/no"
    -a, --abort <string>: <string>: Allowed "Abortion" answers e.g. "q/quit"
  Question:
    <string>            : Display <string> and wait for confirmation.

Examples:
  confirm "Do you want to continue?"
    Result: "Do you want to continue? (y/yes n/no) [no]:"
  confirm "Install the scripts?" -y "y/[yes]"
    Result: "Install the scripts? (y/yes n/no) [yes]:"
  confirm "Sind Sie sicher?" --yes "j/[ja]" --no "n/nein"
    Result: "Sind Sie sicher? (j/ja n/nein) [ja]:"
  confirm "¿Hablas español?" --yes "s/si" --no "n/[no]"
    Result: "¿Hablas español? (s/si n/no) [no]:"
  confirm "Do you want to continue?" --abort "q/quit"
    Result: "Do you want to continue? (y/yes n/no q/quit) [quit]:"

[opc@operator ~]$ confirm "Delete files?" -y "y/[yes]" -n "n/no" -a "q/quit"
Delete files? (y/yes n/no q/quit) [yes]: no
[opc@operator ~]$ echo $?
2
```

Figure 11: Script “confirm”

4.4 Script “transfer”

This script uses "curl" to load a file using the given URL and checks for errors during the download or in the file.

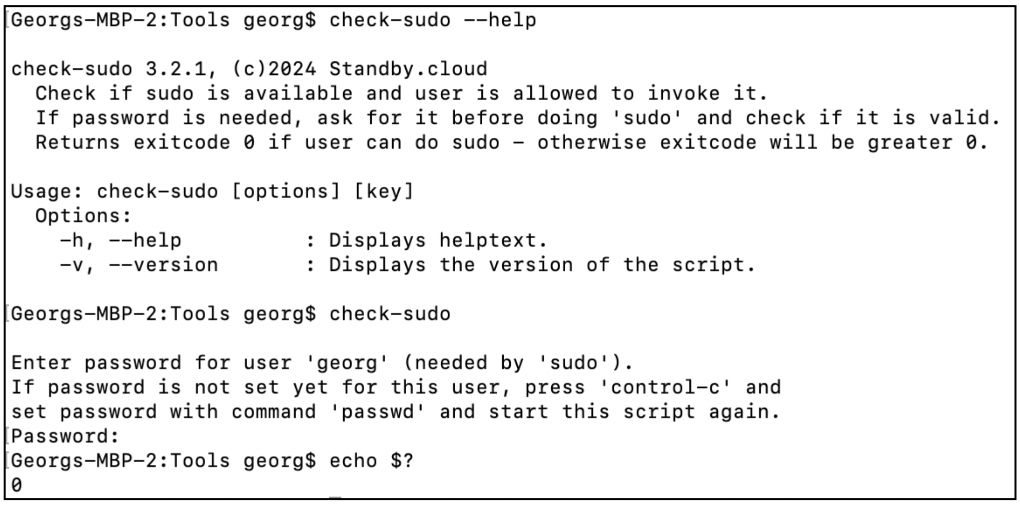


```
[opc@operator ~]$ transfer --help

transfer 3.2.1, (c)2024 Standby.cloud
  Get a resource from a host (or internet) via curl and check if an error occured.

Usage: transfer [options] url
  Options:
    -h, --help             : Displays helptext.
    -v, --version          : Displays the version of the script.
    -q, --quiet            : Don't display any error messages.
    -a, --auth             : Using Oracle Authorization Header e.g. -H "Authorization: Bearer Oracle".
    -s, --seconds <number> : Max <number> of seconds for the operation to take (default is 10).
    -e, --export <filename>: Write output to file "<filename>".
  URL: URL can be specified with or without leading "http://".

Examples:
  transfer http://standby.cloud/download/pdf/Basic-Admin-Scripts.pdf --export manual.pdf
    Downloads manual.pdf from server.
  transfer --auth http://169.254.169.254/opc/v2/instance
    Downloads oci instance metadata for current VM.
```

Figure 12: Script “transfer”

4.5 Script “check-sudo”

This script checks whether the current user can execute commands with root rights using "sudo". If a password is required, this is requested. The return code is 0 if the current user is root or is allowed to execute "sudo". Otherwise, a return code greater than 0 is returned.

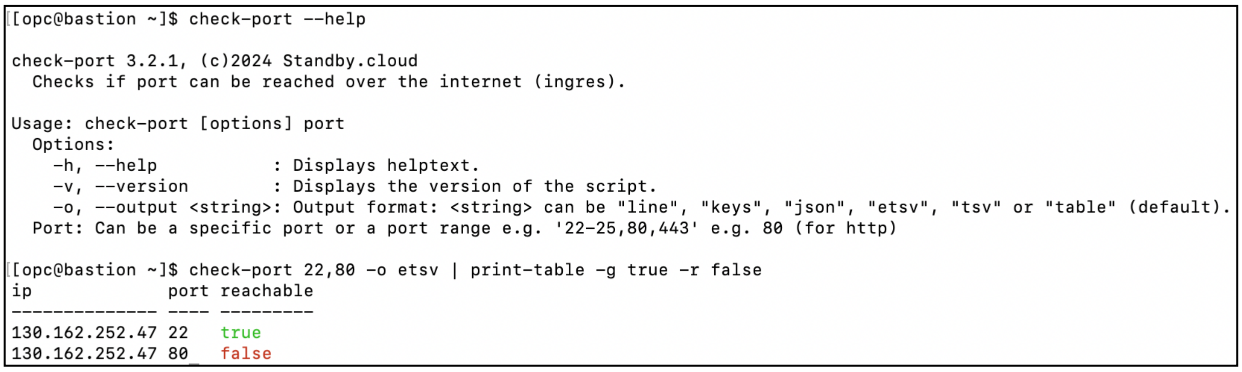


```
Georgs-MBP-2:Tools georg$ check-sudo --help

check-sudo 3.2.1, (c)2024 Standby.cloud
  Check if sudo is available and user is allowed to invoke it.
  If password is needed, ask for it before doing 'sudo' and check if it is valid.
  Returns exitcode 0 if user can do sudo - otherwise exitcode will be greater 0.

Usage: check-sudo [options] [key]
  Options:
    -h, --help          : Displays helptext.
    -v, --version       : Displays the version of the script.

Georgs-MBP-2:Tools georg$ check-sudo

Enter password for user 'georg' (needed by 'sudo').
If password is not set yet for this user, press 'control-c' and
set password with command 'passwd' and start this script again.
Password:
Georgs-MBP-2:Tools georg$ echo $?
0
```

Figure 13: Script “check-sudo”

4.6 Script “check-port”

This script checks whether a port (or a list of ports) is accessible via the Internet. To do this, a connection is established from an Internet server to the current VM (on which the script is running) and checked whether the port is accessible. If the VM cannot be reached from the Internet (e.g., because a firewall blocks this), the result is "false".



```
[opc@bastion ~]$ check-port --help

check-port 3.2.1, (c)2024 Standby.cloud
  Checks if port can be reached over the internet (ingres).

Usage: check-port [options] port
  Options:
    -h, --help          : Displays helptext.
    -v, --version       : Displays the version of the script.
    -o, --output <string>: Output format: <string> can be "line", "keys", "json", "etsv", "tsv" or "table" (default).
  Port: Can be a specific port or a port range e.g. '22-25,80,443' e.g. 80 (for http)

[opc@bastion ~]$ check-port 22,80 -o etsv | print-table -g true -r false
ip             port reachable
-------------- ---- ---------
130.162.252.47 22   true
130.162.252.47 80   false
```

Figure 14: Script “check-port”

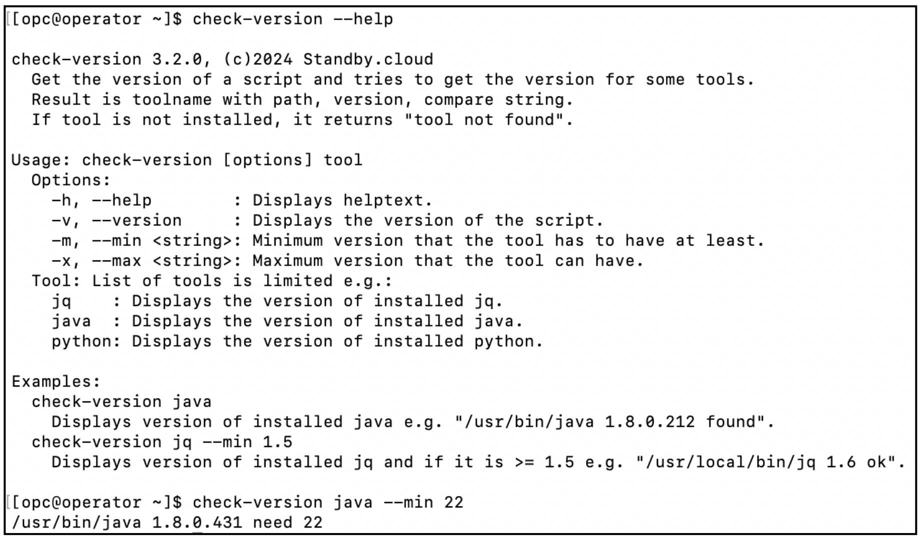
4.7 Script “get-ip”

This script determines the external (public) IP of the instance (or VM) on which the script is executed. A connection is established to an external Internet server, which then tries to reach the instance. If the instance can only be reached via a NAT gateway, for example, the public IP of the NAT gateway is displayed.

Please also read chapter 3.2.1 (help on "get-ip") and chapter 3.5 (return code as a use case for using "get-ip") if you skipped them. In addition to the IP, other key figures are returned, e.g., the approximate location of the IP.

4.8 Script “check-version”

This script checks whether a tool is installed and in which version. First the path is displayed, then the version number and if desired, it can also check whether the version is sufficient (by calling “compare-version”). This script was created primarily for internal use of required tools and versions within the script collection. However, it can also be used for other tools to a limited extent.



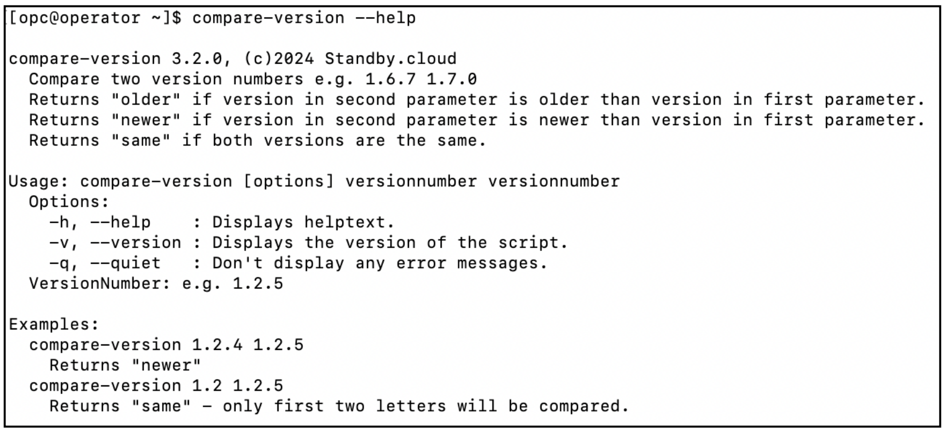
```
[opc@operator ~]$ check-version --help

check-version 3.2.0, (c)2024 Standby.cloud
  Get the version of a script and tries to get the version for some tools.
  Result is toolname with path, version, compare string.
  If tool is not installed, it returns "tool not found".

Usage: check-version [options] tool
  Options:
    -h, --help        : Displays helptext.
    -v, --version     : Displays the version of the script.
    -m, --min <string>: Minimum version that the tool has to have at least.
    -x, --max <string>: Maximum version that the tool can have.
  Tool: List of tools is limited e.g.:
    jq    : Displays the version of installed jq.
    java  : Displays the version of installed java.
    python: Displays the version of installed python.

Examples:
  check-version java
    Displays version of installed java e.g. "/usr/bin/java 1.8.0.212 found".
  check-version jq --min 1.5
    Displays version of installed jq and if it is >= 1.5 e.g. "/usr/local/bin/jq 1.6 ok".

[opc@operator ~]$ check-version java --min 22
/usr/bin/java 1.8.0.431 need 22
```

Figure 15: Script “check-version”

4.9 Script “compare-version”

This script compares two version numbers and returns "older", "newer“, or "same". Example:

```
result=`python --version 2>/dev/null`
stat=$?
if [ $stat -eq 0 ]; then
	curvers=`echo "$result" | cut -d' ' -f2`
	echo "Version '$curvers'."
	result=`compare-version "3.8" "$curvers"`
	if [ "$result" != "newer" ]; then
		echo "Please upgrade to version 3.9 or higher."
	else
		echo "Version sufficient."
	fi
else
	echo "Python not installed."
fi
```

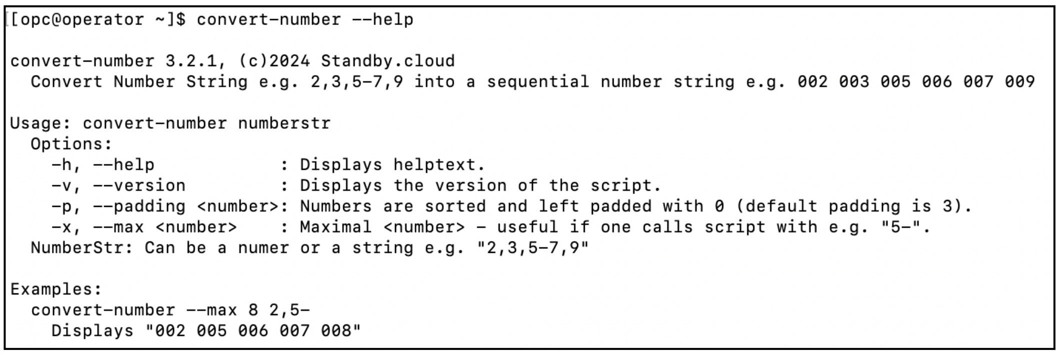


```
[opc@operator ~]$ compare-version --help

compare-version 3.2.0, (c)2024 Standby.cloud
  Compare two version numbers e.g. 1.6.7 1.7.0
  Returns "older" if version in second parameter is older than version in first parameter.
  Returns "newer" if version in second parameter is newer than version in first parameter.
  Returns "same" if both versions are the same.

Usage: compare-version [options] versionnumber versionnumber
  Options:
    -h, --help    : Displays helptext.
    -v, --version : Displays the version of the script.
    -q, --quiet   : Don't display any error messages.
  VersionNumber: e.g. 1.2.5

Examples:
  compare-version 1.2.4 1.2.5
    Returns "newer"
  compare-version 1.2 1.2.5
    Returns "same" - only first two letters will be compared.
```

Figure 16: Script “compare-version”

4.10 Script “convert-number”

This script creates a sequence of numbers based on the passed parameter (numberstr).



```
[opc@operator ~]$ convert-number --help

convert-number 3.2.1, (c)2024 Standby.cloud
  Convert Number String e.g. 2,3,5-7,9 into a sequential number string e.g. 002 003 005 006 007 009

Usage: convert-number numberstr
  Options:
    -h, --help            : Displays helptext.
    -v, --version         : Displays the version of the script.
    -p, --padding <number>: Numbers are sorted and left padded with 0 (default padding is 3).
    -x, --max <number>    : Maximal <number> - useful if one calls script with e.g. "5-".
  NumberStr: Can be a numer or a string e.g. "2,3,5-7,9"

Examples:
  convert-number --max 8 2,5-
    Displays "002 005 006 007 008"
```

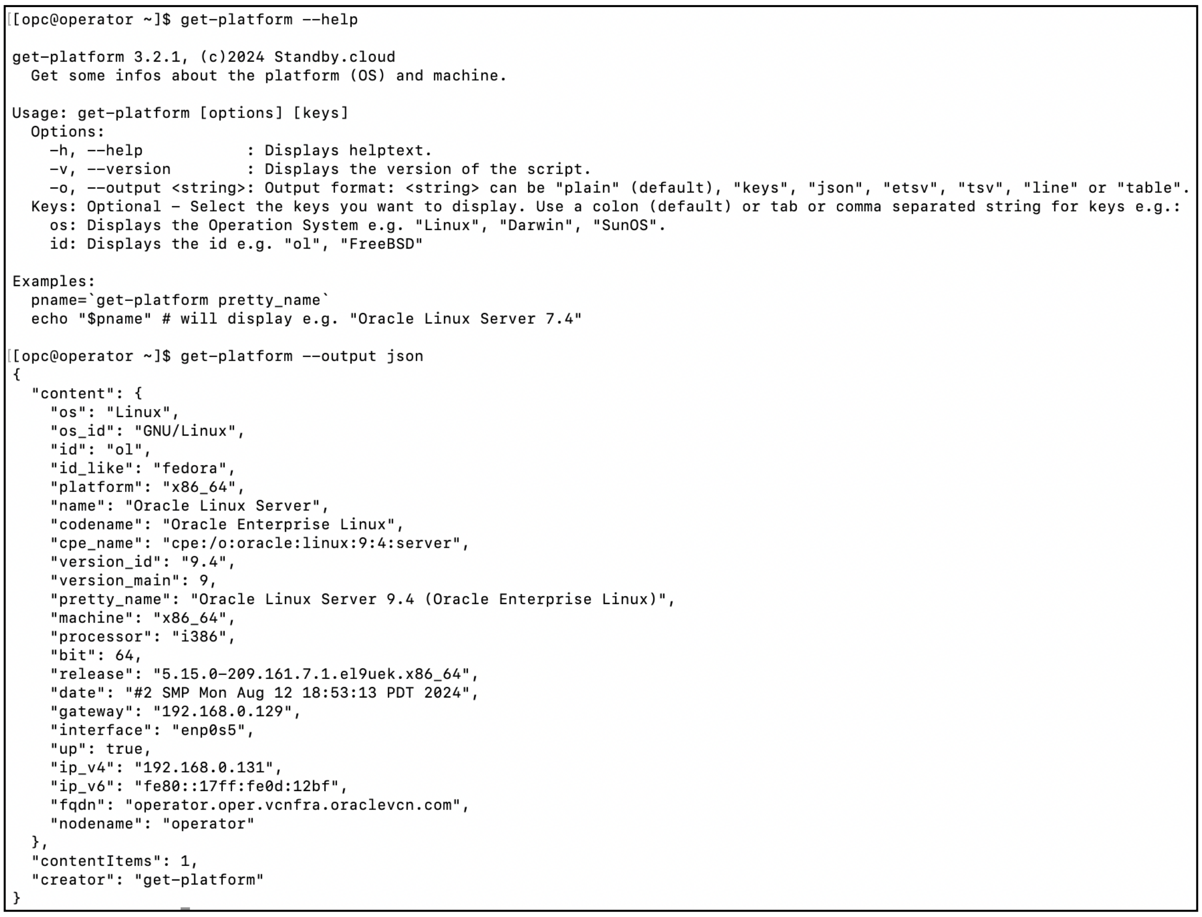
Figure 17: Script “convert-number”

4.11 Script “get-platform”

This script provides detailed information about the platform on which the script runs. Examples of possible return values:

```
      bit: 32 / 64
processor: arm / i386 / sparc
       os: Linux / Darwin (for macOS) / SunOS (for Solaris)
```

Further information about the default gateway and network interface is displayed.

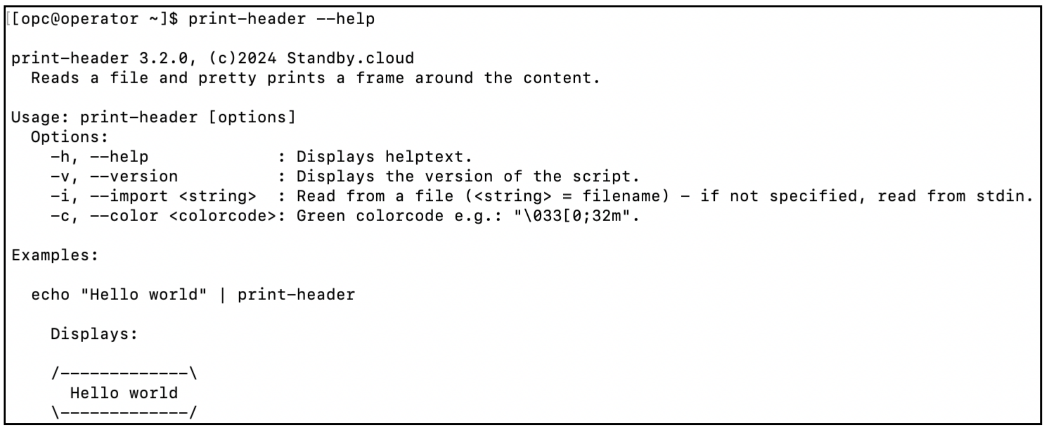


```
[opc@operator ~]$ get-platform --help

get-platform 3.2.1, (c)2024 Standby.cloud
  Get some infos about the platform (OS) and machine.

Usage: get-platform [options] [keys]
  Options:
    -h, --help           : Displays helptext.
    -v, --version        : Displays the version of the script.
    -o, --output <string>: Output format: <string> can be "plain" (default), "keys", "json", "etsv", "tsv", "line" or "table".
  Keys: Optional - Select the keys you want to display. Use a colon (default) or tab or comma separated string for keys e.g.:
    os: Displays the Operation System e.g. "Linux", "Darwin", "SunOS".
    id: Displays the id e.g. "ol", "FreeBSD"

Examples:
  pname=`get-platform pretty_name`
  echo "$pname" # will display e.g. "Oracle Linux Server 7.4"

[opc@operator ~]$ get-platform --output json
{
  "content": {
    "os": "Linux",
    "os_id": "GNU/Linux",
    "id": "ol",
    "id_like": "fedora",
    "platform": "x86_64",
    "name": "Oracle Linux Server",
    "codename": "Oracle Enterprise Linux",
    "cpe_name": "cpe:/o:oracle:linux:9:4:server",
    "version_id": "9.4",
    "version_main": 9,
    "pretty_name": "Oracle Linux Server 9.4 (Oracle Enterprise Linux)",
    "machine": "x86_64",
    "processor": "i386",
    "bit": 64,
    "release": "5.15.0-209.161.7.1.el9uek.x86_64",
    "date": "#2 SMP Mon Aug 12 18:53:13 PDT 2024",
    "gateway": "192.168.0.129",
    "interface": "enp0s5",
    "up": true,
    "ip_v4": "192.168.0.131",
    "ip_v6": "fe80::17ff:fe0d:12bf",
    "fqdn": "operator.oper.vcnfra.oraclevcn.com",
    "nodename": "operator"
  },
  "contentItems": 1,
  "creator": "get-platform"
}
```

Figure 18: Script “get-platform”

4.12 Script “print-header”

This script paints a frame around a heading.

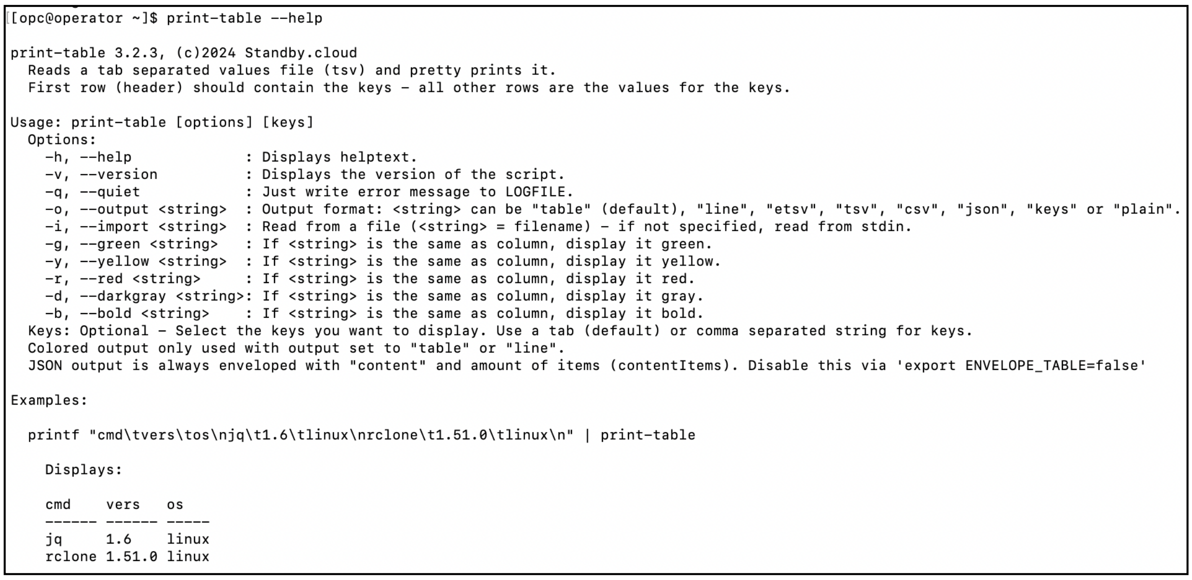


```
[opc@operator ~]$ print-header --help

print-header 3.2.0, (c)2024 Standby.cloud
  Reads a file and pretty prints a frame around the content.

Usage: print-header [options]
  Options:
    -h, --help            : Displays helptext.
    -v, --version         : Displays the version of the script.
    -i, --import <string>  : Read from a file (<string> = filename) - if not specified, read from stdin.
    -c, --color <colorcode>: Green colorcode e.g.: "\033[0;32m".

Examples:

  echo "Hello world" | print-header

    Displays:

    /-------------\
      Hello world
    \-------------/
```

Figure 19: Script “print-header”

4.13 Script “print-table”

This script converts a TSV file into various other formats, e.g., the file can be displayed in table format.



```
[opc@operator ~]$ print-table --help

print-table 3.2.3, (c)2024 Standby.cloud
  Reads a tab separated values file (tsv) and pretty prints it.
  First row (header) should contain the keys - all other rows are the values for the keys.

Usage: print-table [options] [keys]
  Options:
    -h, --help            : Displays helptext.
    -v, --version         : Displays the version of the script.
    -q, --quiet           : Just write error message to LOGFILE.
    -o, --output <string>  : Output format: <string> can be "table" (default), "line", "etsv", "tsv", "csv", "json", "keys" or "plain".
    -i, --import <string>  : Read from a file (<string> = filename) - if not specified, read from stdin.
    -g, --green <string>   : If <string> is the same as column, display it green.
    -y, --yellow <string>  : If <string> is the same as column, display it yellow.
    -r, --red <string>     : If <string> is the same as column, display it red.
    -d, --darkgray <string>: If <string> is the same as column, display it gray.
    -b, --bold <string>    : If <string> is the same as column, display it bold.
  Keys: Optional - Select the keys you want to display. Use a tab (default) or comma separated string for keys.
  Colored output only used with output set to "table" or "line".
  JSON output is always enveloped with "content" and amount of items (contentItems). Disable this via 'export ENVELOPE_TABLE=false'

Examples:

  printf "cmd\tvers\tos\njq\t1.6\tlinux\nrclone\t1.51.0\tlinux\n" | print-table

    Displays:

    cmd    vers   os
    ------ ------ -----
    jq     1.6    linux
    rclone 1.51.0 linux
```

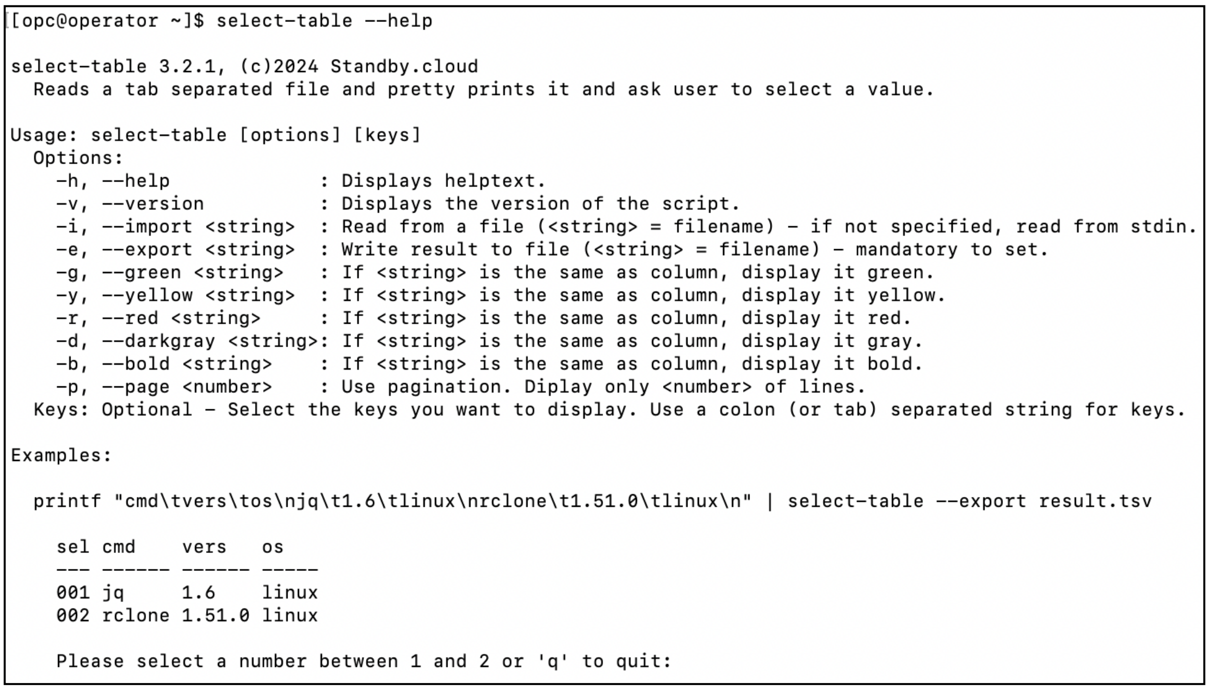
Figure 20: Script “print-table”

The output in JSON is still in an experimental stage.

4.14 Script “select-table”

This script displays (like "print-table") a TSV file in table format (but only this output format is supported here) and then waits for input. A new column "sel" (for Selection) is inserted in front of the table, the rows of which are numbered consecutively.

If you select one of the displayed line numbers, the entire line (without the line number) is written as a return value to the specified output file.



```
[opc@operator ~]$ select-table --help

select-table 3.2.1, (c)2024 Standby.cloud
  Reads a tab separated file and pretty prints it and ask user to select a value.

Usage: select-table [options] [keys]
  Options:
    -h, --help            : Displays helptext.
    -v, --version         : Displays the version of the script.
    -i, --import <string> : Read from a file (<string> = filename) - if not specified, read from stdin.
    -e, --export <string> : Write result to file (<string> = filename) - mandatory to set.
    -g, --green <string>  : If <string> is the same as column, display it green.
    -y, --yellow <string> : If <string> is the same as column, display it yellow.
    -r, --red <string>    : If <string> is the same as column, display it red.
    -d, --darkgray <string>: If <string> is the same as column, display it gray.
    -b, --bold <string>   : If <string> is the same as column, display it bold.
    -p, --page <number>   : Use pagination. Diplay only <number> of lines.
  Keys: Optional - Select the keys you want to display. Use a colon (or tab) separated string for keys.

Examples:

  printf "cmd\tvers\tos\njq\t1.6\tlinux\nrclone\t1.51.0\tlinux\n" | select-table --export result.tsv

    sel cmd    vers   os
    --- ------ ------ -----
    001 jq     1.6    linux
    002 rclone 1.51.0 linux

    Please select a number between 1 and 2 or 'q' to quit:
```

Figure 21: Script "select-table"

4.15 Script "norm-json"

This script checks the syntax of a JSON file for correctness and converts it into a uniform format so that an automatic evaluation can always be carried out according to the same criteria.

Conversion to an array:

An array is always created from the JSON object (or several objects) so that it is easier to access the nth element, for example. You select the nth element in the array with the "--select" option. Specify a number, starting with 1, after the option. For the second element / object, e.g., "--select 2".

Conversion to camelCase (ignored when using the "--raw" option)**:**

All keys in the JSON file are converted to camelCase (https://en.wikipedia.org/wiki/Camel_case) notation. The first letter must be lowercase otherwise the key will not be converted. Example:

`"canonical-region-name" -> "canonicalRegionName"`

Envelopes are stripped (ignored when using the "--raw" option)**:**

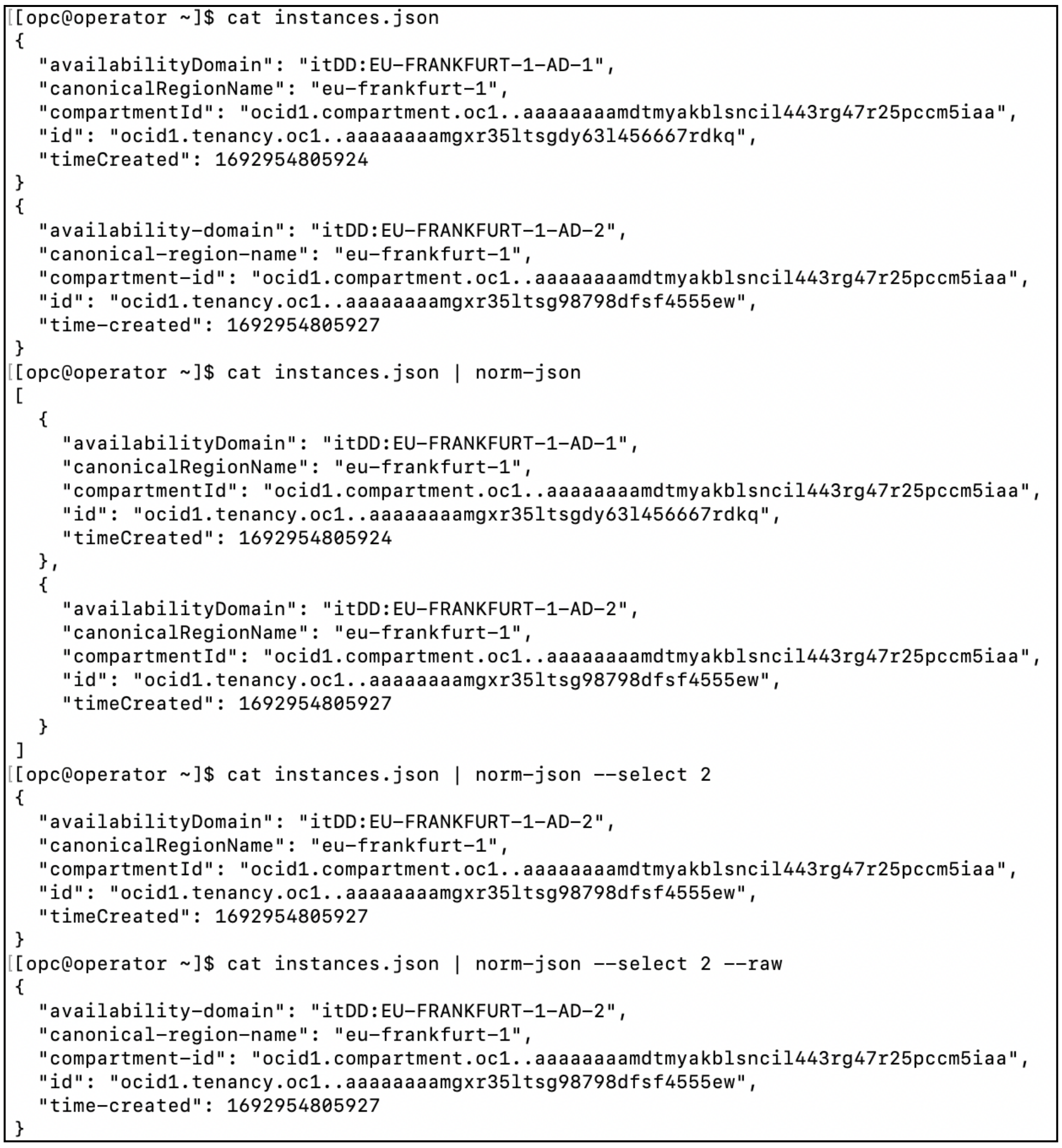
If the JSON was enveloped, it starts with the object "content" (e.g., used by "print-table"), "items" (e.g., used by some object storage resources in OCI), "data" (e.g., used by the tool "oci") or "result" (e.g., used by the tool "opc"), then this enveloping object is removed. Try it yourself with this command: "get-ip -o json" and then enter this command "get-ip -o json | norm-json -s 1".

Comments

If a line begins with "//", the whole line will be removed by the script and can be used for comments. This is an addition to the JSON specification and not listed in the official RFC.

Backslash

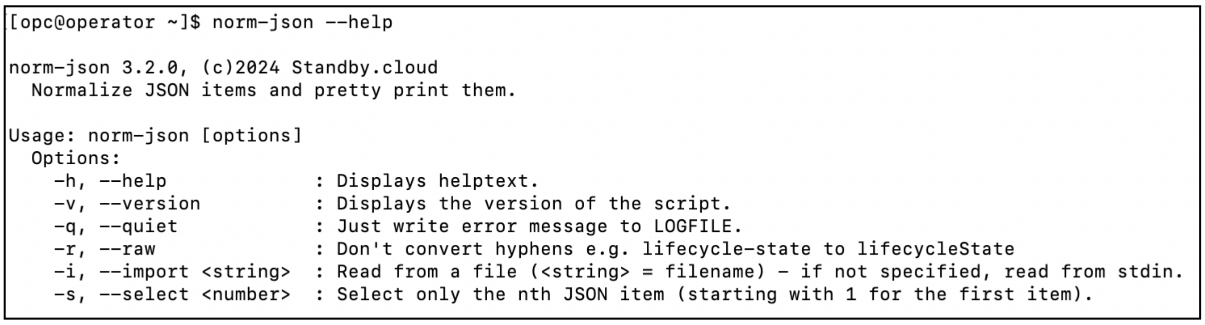
Using a single backslash in JSON is not allowed, but a double backslash (\\) can be used in keys and values although it is unusual. Working with a double backslash can cause confusion because the backslash is masked and is displayed as a single backslash in text output. Therefore, the double backslash is converted to a colon by "norm-json".



```
[[opc@operator ~]$ cat instances.json
{
  "availabilityDomain": "itDD:EU-FRANKFURT-1-AD-1",
  "canonicalRegionName": "eu-frankfurt-1",
  "compartmentId": "ocid1.compartment.oc1..aaaaaaaamdtmyakblsncil443rg47r25pccm5iaa",
  "id": "ocid1.tenancy.oc1..aaaaaaaamgxr35ltsgdy63l456667rdkq",
  "timeCreated": 1692954805924
}
{
  "availability-domain": "itDD:EU-FRANKFURT-1-AD-2",
  "canonical-region-name": "eu-frankfurt-1",
  "compartment-id": "ocid1.compartment.oc1..aaaaaaaamdtmyakblsncil443rg47r25pccm5iaa",
  "id": "ocid1.tenancy.oc1..aaaaaaaamgxr35ltsg98798dfsf4555ew",
  "time-created": 1692954805927
}
[[opc@operator ~]$ cat instances.json | norm-json
[
  {
    "availabilityDomain": "itDD:EU-FRANKFURT-1-AD-1",
    "canonicalRegionName": "eu-frankfurt-1",
    "compartmentId": "ocid1.compartment.oc1..aaaaaaaamdtmyakblsncil443rg47r25pccm5iaa",
    "id": "ocid1.tenancy.oc1..aaaaaaaamgxr35ltsgdy63l456667rdkq",
    "timeCreated": 1692954805924
  },
  {
    "availabilityDomain": "itDD:EU-FRANKFURT-1-AD-2",
    "canonicalRegionName": "eu-frankfurt-1",
    "compartmentId": "ocid1.compartment.oc1..aaaaaaaamdtmyakblsncil443rg47r25pccm5iaa",
    "id": "ocid1.tenancy.oc1..aaaaaaaamgxr35ltsg98798dfsf4555ew",
    "timeCreated": 1692954805927
  }
]
[[opc@operator ~]$ cat instances.json | norm-json --select 2
{
  "availabilityDomain": "itDD:EU-FRANKFURT-1-AD-2",
  "canonicalRegionName": "eu-frankfurt-1",
  "compartmentId": "ocid1.compartment.oc1..aaaaaaaamdtmyakblsncil443rg47r25pccm5iaa",
  "id": "ocid1.tenancy.oc1..aaaaaaaamgxr35ltsg98798dfsf4555ew",
  "timeCreated": 1692954805927
}
[[opc@operator ~]$ cat instances.json | norm-json --select 2 --raw
{
  "availability-domain": "itDD:EU-FRANKFURT-1-AD-2",
  "canonical-region-name": "eu-frankfurt-1",
  "compartment-id": "ocid1.compartment.oc1..aaaaaaaamdtmyakblsncil443rg47r25pccm5iaa",
  "id": "ocid1.tenancy.oc1..aaaaaaaamgxr35ltsg98798dfsf4555ew",
  "time-created": 1692954805927
}
```

Figure 22: Working with “norm-json”

In the above illustration, you see a JSON file with two objects. In the first object, the keys are represented in camelCase notation and in the second object with a hyphen. When the JSON is converted using "norm-json", you get an array with the two objects and both objects use the camelCase notation.

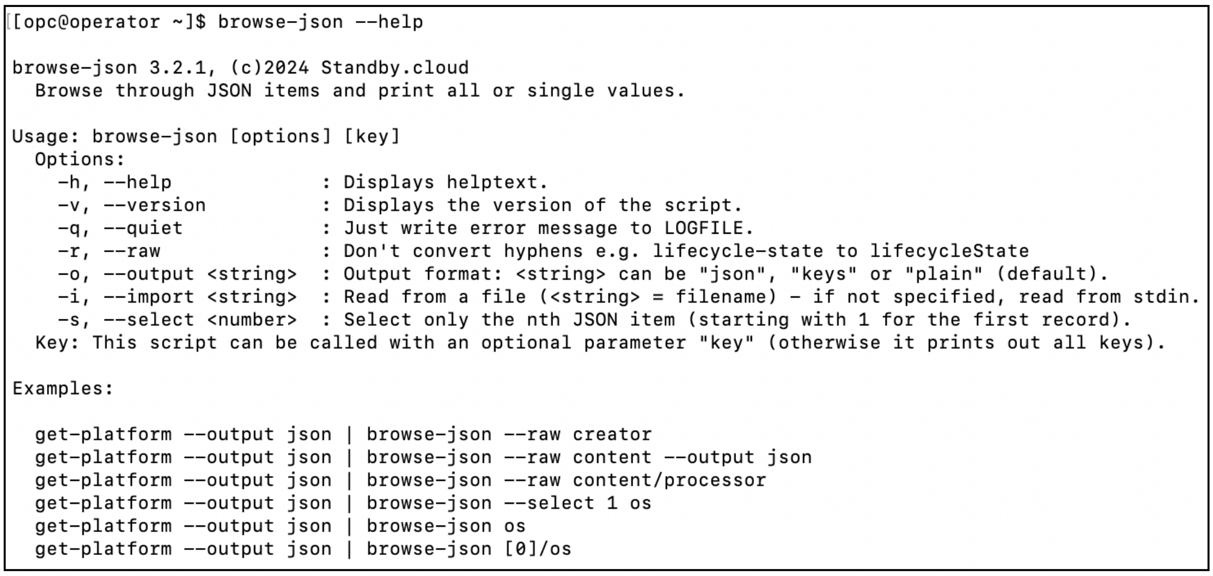


```
[opc@operator ~]$ norm-json --help

norm-json 3.2.0, (c)2024 Standby.cloud
  Normalize JSON items and pretty print them.

Usage: norm-json [options]
  Options:
    -h, --help              : Displays helptext.
    -v, --version           : Displays the version of the script.
    -q, --quiet             : Just write error message to LOGFILE.
    -r, --raw               : Don't convert hyphens e.g. lifecycle-state to lifecycleState
    -i, --import <string>   : Read from a file (<string> = filename) - if not specified, read from stdin.
    -s, --select <number>   : Select only the nth JSON item (starting with 1 for the first item).
```

Figure 23: Script “norm-json”

4.16 Script “browse-json”

This script reads individual values from a JSON object and outputs them in different target formats.

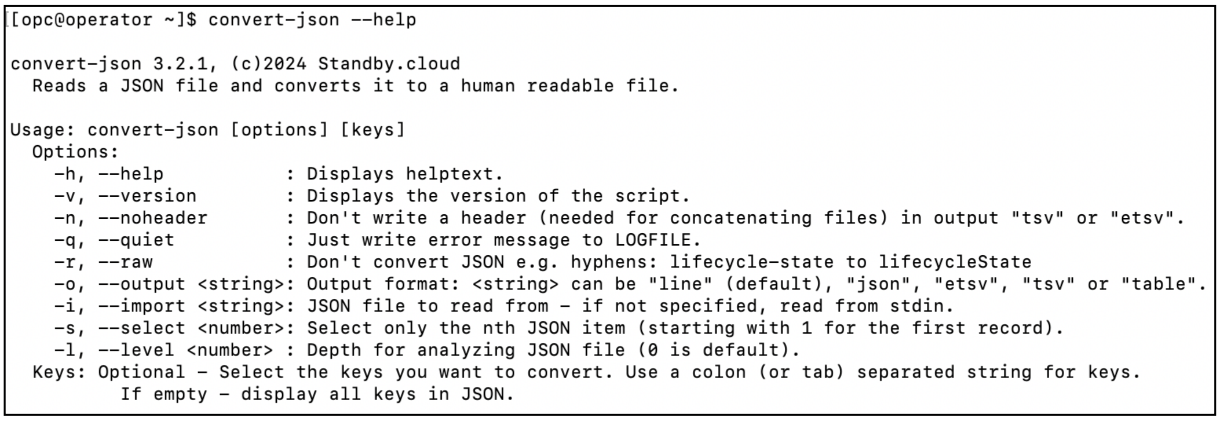


```
[opc@operator ~]$ browse-json --help

browse-json 3.2.1, (c)2024 Standby.cloud
  Browse through JSON items and print all or single values.

Usage: browse-json [options] [key]
  Options:
    -h, --help            : Displays helptext.
    -v, --version         : Displays the version of the script.
    -q, --quiet           : Just write error message to LOGFILE.
    -r, --raw             : Don't convert hyphens e.g. lifecycle-state to lifecycleState
    -o, --output <string> : Output format: <string> can be "json", "keys" or "plain" (default).
    -i, --import <string> : Read from a file (<string> = filename) - if not specified, read from stdin.
    -s, --select <number> : Select only the nth JSON item (starting with 1 for the first record).
  Key: This script can be called with an optional parameter "key" (otherwise it prints out all keys).

Examples:

  get-platform --output json | browse-json --raw creator
  get-platform --output json | browse-json --raw content --output json
  get-platform --output json | browse-json --raw content/processor
  get-platform --output json | browse-json --select 1 os
  get-platform --output json | browse-json os
  get-platform --output json | browse-json [0]/os
```

Figure 24: Script “browse-json”

4.17 Script “convert-json”

This script reads all values from a JSON object (at a specified depth) and outputs them in different target formats.



```
[opc@operator ~]$ convert-json --help

convert-json 3.2.1, (c)2024 Standby.cloud
  Reads a JSON file and converts it to a human readable file.

Usage: convert-json [options] [keys]
  Options:
    -h, --help           : Displays helptext.
    -v, --version        : Displays the version of the script.
    -n, --noheader       : Don't write a header (needed for concatenating files) in output "tsv" or "etsv".
    -q, --quiet          : Just write error message to LOGFILE.
    -r, --raw            : Don't convert JSON e.g. hyphens: lifecycle-state to lifecycleState
    -o, --output <string>: Output format: <string> can be "line" (default), "json", "etsv", "tsv" or "table".
    -i, --import <string>: JSON file to read from - if not specified, read from stdin.
    -s, --select <number>: Select only the nth JSON item (starting with 1 for the first record).
    -l, --level <number> : Depth for analyzing JSON file (0 is default).
  Keys: Optional - Select the keys you want to convert. Use a colon (or tab) separated string for keys.
          If empty - display all keys in JSON.
```

Figure 25: Script “convert-json”

5 SUMMARY

With this document, you received an overview of the available cross-platform scripts (written in BASH), which can be used, modified, and used in your own scripts at any time free of charge.