jc

CLI tool and python library that converts the output of popular command-line tools, file-types, and common strings to JSON, YAML, or Dictionaries. This allows piping of output to tools like jq and simplifying automation scripts.

View on GitHub

Tests passing pypi v1.22.3

> Check out the `jc` Python package documentation for developers

> Try the `jc` web demo

> JC is now available as an Ansible filter plugin in the `community.general` collection. See this blog post for an example.

JC

JSON Convert

`jc` JSONifies the output of many CLI tools, file-types, and common strings for easier parsing in scripts. See the **Parsers** section for supported commands, file-types, and strings.

```
dig example.com | jc --dig
```

```
[{"id":38052,"opcode":"QUERY","status":"NOERROR","flags":["qr","rd","ra"],
"query_num":1,"answer_num":1,"authority_num":0,"additional_num":1,
"opt_pseudosection":{"edns":{"version":0,"flags":[],"udp":4096}},"question":
{"name":"example.com.","class":"IN","type":"A"},"answer":[{"name":
"example.com.","class":"IN","type":"A","ttl":39049,"data":"93.184.216.34"}],
"query_time":49,"server":"2600:1700:bab0:d40::1#53(2600:1700:bab0:d40::1)",
"when":"Fri Apr 16 16:09:00 PDT 2021","rcvd":56,"when_epoch":1618614540,
"when_epoch_utc":null}]
```

This allows further command-line processing of output with tools like `jq` or `jello` by piping commands:

```
$ dig example.com | jc --dig | jq -r '.[].answer[].data'
93.184.216.34
```

or using the alternative "magic" syntax:

```
$ jc dig example.com | jq -r '.[].answer[].data'
93.184.216.34
```

`jc` can also be used as a python library. In this case the output will be a python dictionary, a list of dictionaries, or even a lazy iterable of dictionaries instead of JSON:

```
>>> import subprocess
>>> import jc
>>>
>>> cmd_output = subprocess.check_output(['dig', 'example.com'], text=True)
>>> data = jc.parse('dig', cmd_output)
>>>
>>> data[0]['answer']
[{'name': 'example.com.', 'class': 'IN', 'type': 'A', 'ttl': 29658, 'data':
'93.184.216.34'}]
```

> For `jc` Python package documentation, use `help('jc')`, `help('jc.lib')`, or see the online documentation.

Two representations of the data are available. The default representation uses a strict schema per parser and converts known numbers to int/float JSON values. Certain known values of `None` are converted to JSON `null`, known boolean values are converted, and, in some cases, additional semantic context fields are added.

To access the raw, pre-processed JSON, use the `-r` cli option or the `raw=True` function parameter in `parse()` when using `jc` as a python library.

Schemas for each parser can be found at the documentation link beside each **Parser** below.

Release notes can be found here.

Why Would Anyone Do This!?

For more information on the motivations for this project, please see my blog post on Bringing the Unix Philosophy to the 21st Century and my interview with Console.

See also:

- libxo on FreeBSD
- powershell
- blog: linux apps should have a json flag
- Hacker News discussion
- Reddit discussion

Use Cases:

- Bash scripting
- Ansible command output parsing
- Saltstack command output parsing
- Nornir command output parsing
- FortiSOAR command output parsing

Installation

There are several ways to get `jc`. You can install via `pip`, OS package repositories, or by downloading the correct binary for your architecture and running it anywhere on your filesystem.

Pip (macOS, linux, unix, Windows)

pypi v1.22.3

```
pip3 install jc
```

OS Package Repositories

OS	Command
Debian/Ubuntu linux	`apt-get install jc`
Fedora linux	`dnf install jc`
openSUSE linux	`zypper install jc`
Arch linux	`pacman -S jc`
NixOS linux	`nix-env -iA nixpkgs.jc` or `nix-env -iA nixos.jc`
Guix System linux	`guix install jc`
Gentoo Linux	`emerge dev-python/jc`
macOS	`brew install jc`
FreeBSD	`portsnap fetch update && cd /usr/ports/textproc/py-jc && make install clean`
Ansible filter plugin	`ansible-galaxy collection install community.general`

OS	Command
FortiSOAR connector	Install from FortiSOAR Connector Marketplace

> For more OS Packages, see https://repology.org/project/jc/versions.

Binaries

For precompiled binaries, see Releases on Github.

Usage

`jc` accepts piped input from `STDIN` and outputs a JSON representation of the previous command's output to `STDOUT`.

```
COMMAND | jc [OPTIONS] PARSER
cat FILE | jc [OPTIONS] PARSER
echo STRING | jc [OPTIONS] PARSER
```

Alternatively, the "magic" syntax can be used by prepending `jc` to the command to be converted or in front of the absolute path for Proc files. Options can be passed to `jc` immediately before the command or Proc file path is given. (Note: command aliases and shell builtins are not supported)

```
jc [OPTIONS] COMMAND
jc [OPTIONS] /proc/<path-to-procfile>
```

The JSON output can be compact (default) or pretty formatted with the `-p` option.

Parsers

Argument	Command or Filetype	Documentation
`--acpi`	`acpi` command parser	details
`--airport`	`airport -I` command parser	details
`--airport-s`	`airport -s` command parser	details
`--arp`	`arp` command parser	details
`--asciitable`	ASCII and Unicode table parser	details

Argument	Command or Filetype	Documentation
`--asciitable-m`	multi-line ASCII and Unicode table parser	details
`--blkid`	`blkid` command parser	details
`--cbt`	`cbt` (Google Bigtable) command parser	details
`--cef`	CEF string parser	details
`--cef-s`	CEF string streaming parser	details
`--chage`	`chage --list` command parser	details
`--cksum`	`cksum` and `sum` command parser	details
`--clf`	Common and Combined Log Format file parser	details
`--clf-s`	Common and Combined Log Format file streaming parser	details
`--crontab`	`crontab` command and file parser	details
`--crontab-u`	`crontab` file parser with user support	details
`--csv`	CSV file parser	details
`--csv-s`	CSV file streaming parser	details
`--date`	`date` command parser	details
`--datetime-iso`	ISO 8601 Datetime string parser	details
`--df`	`df` command parser	details
`--dig`	`dig` command parser	details
`--dir`	`dir` command parser	details
`--dmidecode`	`dmidecode` command parser	details
`--dpkg-l`	`dpkg -l` command parser	details
`--du`	`du` command parser	details
`--email-address`	Email Address string parser	details
`--env`	`env` command parser	details

Argument	Command or Filetype	Documentation
`--file`	`file` command parser	details
`--findmnt`	`findmnt` command parser	details
`--finger`	`finger` command parser	details
`--free`	`free` command parser	details
`--fstab`	`/etc/fstab` file parser	details
`--git-log`	`git log` command parser	details
`--git-log-s`	`git log` command streaming parser	details
`--git-ls-remote`	`git ls-remote` command parser	details
`--gpg`	`gpg --with-colons` command parser	details
`--group`	`/etc/group` file parser	details
`--gshadow`	`/etc/gshadow` file parser	details
`--hash`	`hash` command parser	details
`--hashsum`	hashsum command parser (`md5sum` , `shasum` , etc.)	details
`--hciconfig`	`hciconfig` command parser	details
`--history`	`history` command parser	details
`--hosts`	`/etc/hosts` file parser	details
`--id`	`id` command parser	details
`--ifconfig`	`ifconfig` command parser	details
`--ini`	INI file parser	details
`--iostat`	`iostat` command parser	details
`--iostat-s`	`iostat` command streaming parser	details
`--ip-address`	IPv4 and IPv6 Address string parser	details
`--iptables`	`iptables` command parser	details

Argument	Command or Filetype	Documentation
`--iw-scan`	`iw dev [device] scan` command parser	details
`--jar-manifest`	Java MANIFEST.MF file parser	details
`--jobs`	`jobs` command parser	details
`--jwt`	JWT string parser	details
`--kv`	Key/Value file parser	details
`--last`	`last` and `lastb` command parser	details
`--ls`	`ls` command parser	details
`--ls-s`	`ls` command streaming parser	details
`--lsblk`	`lsblk` command parser	details
`--lsmod`	`lsmod` command parser	details
`--lsof`	`lsof` command parser	details
`--lspci`	`lspci -mmv` command parser	details
`--lsusb`	`lsusb` command parser	details
`--m3u`	M3U and M3U8 file parser	details
`--mdadm`	`mdadm` command parser	details
`--mount`	`mount` command parser	details
`--mpstat`	`mpstat` command parser	details
`--mpstat-s`	`mpstat` command streaming parser	details
`--netstat`	`netstat` command parser	details
`--nmcli`	`nmcli` command parser	details
`--ntpq`	`ntpq -p` command parser	details
`--openvpn`	openvpn-status.log file parser	details
`--os-prober`	`os-prober` command parser	details

Argument	Command or Filetype	Documentation
`--passwd`	`/etc/passwd` file parser	details
`--pci-ids`	`pci.ids` file parser	details
`--pgpass`	PostgreSQL password file parser	details
`--pidstat`	`pidstat -H` command parser	details
`--pidstat-s`	`pidstat -H` command streaming parser	details
`--ping`	`ping` and `ping6` command parser	details
`--ping-s`	`ping` and `ping6` command streaming parser	details
`--pip-list`	`pip list` command parser	details
`--pip-show`	`pip show` command parser	details
`--plist`	PLIST file parser	details
`--postconf`	`postconf -M` command parser	details
`--proc`	`/proc/` file parser	details
`--ps`	`ps` command parser	details
`--route`	`route` command parser	details
`--rpm-qi`	`rpm -qi` command parser	details
`--rsync`	`rsync` command parser	details
`--rsync-s`	`rsync` command streaming parser	details
`--semver`	Semantic Version string parser	details
`--sfdisk`	`sfdisk` command parser	details
`--shadow`	`/etc/shadow` file parser	details
`--ss`	`ss` command parser	details
`--sshd-conf`	sshd config file and `sshd -T` command parser	details
`--stat`	`stat` command parser	details

Argument	Command or Filetype	Documentation
`--stat-s`	`stat` command streaming parser	details
`--sysctl`	`sysctl` command parser	details
`--syslog`	Syslog RFC 5424 string parser	details
`--syslog-s`	Syslog RFC 5424 string streaming parser	details
`--syslog-bsd`	Syslog RFC 3164 string parser	details
`--syslog-bsd-s`	Syslog RFC 3164 string streaming parser	details
`--systemctl`	`systemctl` command parser	details
`--systemctl-lj`	`systemctl list-jobs` command parser	details
`--systemctl-ls`	`systemctl list-sockets` command parser	details
`--systemctl-luf`	`systemctl list-unit-files` command parser	details
`--systeminfo`	`systeminfo` command parser	details
`--time`	`/usr/bin/time` command parser	details
`--timedatectl`	`timedatectl status` command parser	details
`--timestamp`	Unix Epoch Timestamp string parser	details
`--top`	`top -b` command parser	details
`--top-s`	`top -b` command streaming parser	details
`--tracepath`	`tracepath` and `tracepath6` command parser	details
`--traceroute`	`traceroute` and `traceroute6` command parser	details
`--udevadm`	`udevadm info` command parser	details
`--ufw`	`ufw status` command parser	details
`--ufw-appinfo`	`ufw app info [application]` command parser	details
`--uname`	`uname -a` command parser	details

Argument	Command or Filetype	Documentation
`--update-alt-gs`	`update-alternatives --get-selections` command parser	details
`--update-alt-q`	`update-alternatives --query` command parser	details
`--upower`	`upower` command parser	details
`--uptime`	`uptime` command parser	details
`--url`	URL string parser	details
`--vmstat`	`vmstat` command parser	details
`--vmstat-s`	`vmstat` command streaming parser	details
`--w`	`w` command parser	details
`--wc`	`wc` command parser	details
`--who`	`who` command parser	details
`--x509-cert`	X.509 PEM and DER certificate file parser	details
`--xml`	XML file parser	details
`--xrandr`	`xrandr` command parser	details
`--yaml`	YAML file parser	details
`--zipinfo`	`zipinfo` command parser	details

Options

Short	Long	Description
`-a`	`--about`	About `jc`. Prints information about `jc` and the parsers (in JSON or YAML, of course!)
`-C`	`--force-color`	Force color output even when using pipes (overrides `-m` and the `NO_COLOR` env variable)
`-d`	`--debug`	Debug mode. Prints trace messages if parsing issues are encountered (use `-dd` for verbose debugging)

Short	Long	Description
`-h`	`--help`	Help. Use `jc -h --parser_name` for parser documentation. Use twice to show hidden parsers (e.g. `-hh`)
`-m`	`--monochrome`	Monochrome output
`-M`	`--meta-out`	Add metadata to output including timestamp, parser name, magic command, magic command exit code, etc.
`-p`	`--pretty`	Pretty format the JSON output
`-q`	`--quiet`	Quiet mode. Suppresses parser warning messages (use `-qq` to ignore streaming parser errors)
`-r`	`--raw`	Raw output. Provides more literal output, typically with string values and no additional semantic processing
`-u`	`--unbuffer`	Unbuffer output
`-v`	`--version`	Version information
`-y`	`--yaml-out`	YAML output
`-B`	`--bash-comp`	Generate Bash shell completion script (more info)
`-Z`	`--zsh-comp`	Generate Zsh shell completion script (more info)

Exit Codes

Any fatal errors within `jc` will generate an exit code of `100` , otherwise the exit code will be `0` .

When using the "magic" syntax (e.g. `jc ifconfig eth0`), `jc` will store the exit code of the program being parsed and add it to the `jc` exit code. This way it is easier to determine if an error was from the parsed program or `jc` .

Consider the following examples using `ifconfig` :

`ifconfig` exit code	`jc` exit code	Combined exit code	Interpretation
`0`	`0`	`0`	No errors
`1`	`0`	`1`	Error in `ifconfig`
`0`	`100`	`100`	Error in `jc`

`ifconfig` exit code	`jc` exit code	Combined exit code	Interpretation
`1`	`100`	`101`	Error in both `ifconfig` and `jc`

When using the "magic" syntax you can also retrieve the exit code of the called program by using the `--meta-out` or `-M` option. This will append a `_jc_meta` object to the output that will include the magic command information, including the exit code.

Here is an example with `ping`:

```
$ jc --meta-out -p ping -c2 192.168.1.252
{
  "destination_ip": "192.168.1.252",
  "data_bytes": 56,
  "pattern": null,
  "destination": "192.168.1.252",
  "packets_transmitted": 2,
  "packets_received": 0,
  "packet_loss_percent": 100.0,
  "duplicates": 0,
  "responses": [
    {
      "type": "timeout",
      "icmp_seq": 0,
      "duplicate": false
    }
  ],
  "_jc_meta": {
    "parser": "ping",
    "timestamp": 1661357115.27949,
    "magic_command": [
      "ping",
      "-c2",
      "192.168.1.252"
    ],
    "magic_command_exit": 2
  }
}
$ echo $?
2
```

Setting Custom Colors via Environment Variable

You can specify custom colors via the `JC_COLORS` environment variable. The `JC_COLORS` environment variable takes four comma separated string values in the following format:

```
JC_COLORS=<keyname_color>,<keyword_color>,<number_color>,<string_color>
```

Where colors are: `black`, `red`, `green`, `yellow`, `blue`, `magenta`, `cyan`, `gray`, `brightblack`, `brightred`, `brightgreen`, `brightyellow`, `brightblue`, `brightmagenta`, `brightcyan`, `white`, or `default`

For example, to set to the default colors:

```
JC_COLORS=blue,brightblack,magenta,green
```

or

```
JC_COLORS=default,default,default,default
```

Disable Colors via Environment Variable

You can set the `NO_COLOR` environment variable to any value to disable color output in `jc`. Note that using the `-C` option to force color output will override both the `NO_COLOR` environment variable and the `-m` option.

Streaming Parsers

Most parsers load all of the data from `STDIN`, parse it, then output the entire JSON document serially. There are some streaming parsers (e.g. `ls-s` and `ping-s`) that immediately start processing and outputting the data line-by-line as JSON Lines (aka NDJSON) while it is being received from `STDIN`. This can significantly reduce the amount of memory required to parse large amounts of command output (e.g. `ls -lR /`) and can sometimes process the data more quickly. Streaming parsers have slightly different behavior than standard parsers as outlined below.

> Note: Streaming parsers cannot be used with the "magic" syntax

Ignoring Errors

You may want to ignore parsing errors when using streaming parsers since these may be used in long-lived processing pipelines and errors can break the pipe. To ignore parsing errors, use the `-qq` cli option or the `ignore_exceptions=True` argument with the `parse()` function. This will add a `_jc_meta` object to the JSON output with a `success` attribute. If `success` is `true`, then there were no issues parsing the line. If `success` is `false`, then a parsing issue was found and `error` and `line` fields will be added to include a short error description and the contents of the unparsable line, respectively:

Successfully parsed line with `-qq` option:

```
{
  "command_data": "data",
  "_jc_meta": {
    "success": true
  }
}
```

Unsuccessfully parsed line with `-qq` option:

```
{
  "_jc_meta": {
    "success": false,
    "error": "error message",
    "line": "original line data"
  }
}
```

Unbuffering Output

Most operating systems will buffer output that is being piped from process to process. The buffer is usually around 4KB. When viewing the output in the terminal the OS buffer is not engaged so output is immediately displayed on the screen. When piping multiple processes together, though, it may seem as if the output is hanging when the input data is very slow (e.g. `ping`):

```
$ ping 1.1.1.1 | jc --ping-s | jq
<slow output>
```

This is because the OS engages the 4KB buffer between `jc` and `jq` in this example. To display the data on the terminal in realtime, you can disable the buffer with the `-u` (unbuffer) cli option:

```
$ ping 1.1.1.1 | jc --ping-s -u | jq
{"type":"reply","pattern":null,"timestamp":null,"bytes":"64","respons...}
{"type":"reply","pattern":null,"timestamp":null,"bytes":"64","respons...}
...
```

> Note: Unbuffered output can be slower for large data streams.

Using Streaming Parsers as Python Modules

Streaming parsers accept any iterable object and return an iterable object allowing lazy processing of the data. The input data should iterate on lines of string data. Examples of good input data are `sys.stdin` or `str.splitlines()` .

To use the returned iterable object in your code, simply loop through it or use the next() builtin function:

```
import jc

result = jc.parse('ls_s', ls_command_output.splitlines())
for item in result:
    print(item["filename"])
```

Custom Parsers

Custom local parser plugins may be placed in a `jc/jcparsers` folder in your local **"App data directory"**:

- Linux/unix: `$HOME/.local/share/jc/jcparsers`
- macOS: `$HOME/Library/Application Support/jc/jcparsers`
- Windows: `$LOCALAPPDATA\jc\jc\jcparsers`

Local parser plugins are standard python module files. Use the `jc/parsers/foo.py` or `jc/parsers/foo_s.py (streaming)` parser as a template and simply place a `.py` file in the `jcparsers` subfolder.

Local plugin filenames must be valid python module names and therefore must start with a letter and consist entirely of alphanumerics and underscores. Local plugins may override default parsers.

> Note: The application data directory follows the XDG Base Directory Specification

Caveats

Locale

For best results set the locale environment variables to `C` or `en_US.UTF-8` by modifying the `LC_ALL` variable:

```
$ LC_ALL=C date | jc --date
```

You can also set the locale variables individually:

```
$ export LANG=C
$ export LC_NUMERIC=C
```

On some older systems UTF-8 output will be downgraded to ASCII with `\\u` escape sequences if the `C` locale does not support UTF-8 encoding.

Timezones

Some parsers have calculated epoch timestamp fields added to the output. Unless a timestamp field name has a `_utc` suffix it is considered naive. (i.e. based on the local timezone of the system the `jc` parser was run on).

If a UTC timezone can be detected in the text of the command output, the timestamp will be timezone aware and have a `_utc` suffix on the key name. (e.g. `epoch_utc`) No other timezones are supported for aware timestamps.

Use In Other Shells

`jc` can be used in most any shell. Some modern shells have JSON deserialization and filtering capabilities built-in which makes using `jc` even more convenient.

For example, the following is possible in NGS (Next Generation Shell):

```
myvar = ``jc dig www.google.com``[0].answer[0].data
```

This runs `jc`, parses the output JSON, and assigs the resulting data structure to a variable in a single line of code.

For more examples of how to use `jc` in other shells, see this wiki page.

Compatibility

Some parsers like `dig`, `xml`, `csv`, etc. will work on any platform. Other parsers that convert platform-specific output will generate a warning message if they are run on an unsupported platform. To see all parser information, including compatibility, run `jc -ap`.

You may still use a parser on an unsupported platform - for example, you may want to parse a file with linux `lsof` output on an macOS or Windows laptop. In that case you can suppress the warning message with the `-q` cli option or the `quiet=True` function parameter in `parse()`:

macOS:

```
cat lsof.out | jc -q --lsof
```

or Windows:

```
type lsof.out | jc -q --lsof
```

Tested on:

- Centos 7.7

- Ubuntu 18.04
- Ubuntu 20.04
- Fedora32
- macOS 10.11.6
- macOS 10.14.6
- NixOS
- FreeBSD12
- Windows 10
- Windows 2016 Server
- Windows 2019 Server

Contributions

Feel free to add/improve code or parsers! You can use the `jc/parsers/foo.py` or `jc/parsers/foo_s.py (streaming)` parsers as a template and submit your parser with a pull request.

Please see the Contributing Guidelines for more information.

Acknowledgments

- Local parser plugin feature contributed by Dean Serenevy
- CI automation and code optimizations by philippeitis
- `ifconfig-parser` module by KnightWhoSayNi
- `xmltodict` module by Martín Blech
- `ruamel.yaml` module by Anthon van der Neut
- `trparse` module by Luis Benitez
- Parsing code from Conor Heine adapted for some parsers
- Excellent constructive feedback from Ilya Sher

Examples

Here are some examples of `jc` output. For more examples, see here or the parser documentation.

arp

```
arp | jc -p --arp          # or:  jc -p arp
```

```
[
  {
    "address": "gateway",
    "hwtype": "ether",
    "hwaddress": "00:50:56:f7:4a:fc",
    "flags_mask": "C",
    "iface": "ens33"
  },
  {
    "address": "192.168.71.1",
    "hwtype": "ether",
    "hwaddress": "00:50:56:c0:00:08",
    "flags_mask": "C",
    "iface": "ens33"
  },
  {
    "address": "192.168.71.254",
    "hwtype": "ether",
    "hwaddress": "00:50:56:fe:7a:b4",
    "flags_mask": "C",
    "iface": "ens33"
  }
]
```

CSV files

```
cat homes.csv
```

```
"Sell", "List", "Living", "Rooms", "Beds", "Baths", "Age", "Acres", "Taxes"
142, 160, 28, 10, 5, 3,  60, 0.28,  3167
175, 180, 18,  8, 4, 1,  12, 0.43,  4033
129, 132, 13,  6, 3, 1,  41, 0.33,  1471
...
```

```
cat homes.csv | jc -p --csv
```

```
[
  {
    "Sell": "142",
    "List": "160",
    "Living": "28",
    "Rooms": "10",
    "Beds": "5",
    "Baths": "3",
    "Age": "60",
    "Acres": "0.28",
    "Taxes": "3167"
  },
  {
    "Sell": "175",
    "List": "180",
    "Living": "18",
    "Rooms": "8",
    "Beds": "4",
    "Baths": "1",
    "Age": "12",
    "Acres": "0.43",
    "Taxes": "4033"
  },
  {
    "Sell": "129",
    "List": "132",
    "Living": "13",
    "Rooms": "6",
    "Beds": "3",
    "Baths": "1",
    "Age": "41",
    "Acres": "0.33",
    "Taxes": "1471"
  }
]
```

/etc/hosts file

```
cat /etc/hosts | jc -p --hosts
```

```
[
  {
    "ip": "127.0.0.1",
    "hostname": [
      "localhost"
    ]
  },
  {
    "ip": "::1",
    "hostname": [
      "ip6-localhost",
      "ip6-loopback"
    ]
  },
  {
    "ip": "fe00::0",
    "hostname": [
      "ip6-localnet"
    ]
  }
]
```

ifconfig

```
ifconfig | jc -p --ifconfig          # or:  jc -p ifconfig
```

```
[
  {
    "name": "ens33",
    "flags": 4163,
    "state": [
      "UP",
      "BROADCAST",
      "RUNNING",
      "MULTICAST"
    ],
    "mtu": 1500,
    "ipv4_addr": "192.168.71.137",
    "ipv4_mask": "255.255.255.0",
    "ipv4_bcast": "192.168.71.255",
    "ipv6_addr": "fe80::c1cb:715d:bc3e:b8a0",
    "ipv6_mask": 64,
    "ipv6_scope": "0x20",
    "mac_addr": "00:0c:29:3b:58:0e",
    "type": "Ethernet",
    "rx_packets": 8061,
    "rx_bytes": 1514413,
    "rx_errors": 0,
    "rx_dropped": 0,
    "rx_overruns": 0,
    "rx_frame": 0,
    "tx_packets": 4502,
    "tx_bytes": 866622,
    "tx_errors": 0,
    "tx_dropped": 0,
    "tx_overruns": 0,
    "tx_carrier": 0,
    "tx_collisions": 0,
    "metric": null
  }
]
```

INI files

```
cat example.ini
```

```
[DEFAULT]
ServerAliveInterval = 45
Compression = yes
CompressionLevel = 9
ForwardX11 = yes

[bitbucket.org]
User = hg

[topsecret.server.com]
Port = 50022
ForwardX11 = no
```

```
cat example.ini | jc -p --ini
```

```
{
  "bitbucket.org": {
    "ServeraLiveInterval": "45",
    "Compression": "yes",
    "CompressionLevel": "9",
    "ForwardX11": "yes",
    "User": "hg"
  },
  "topsecret.server.com": {
    "ServeraLiveInterval": "45",
    "Compression": "yes",
    "CompressionLevel": "9",
    "ForwardX11": "no",
    "Port": "50022"
  }
}
```

ls

```
$ ls -l /usr/bin | jc -p --ls          # or:  jc -p ls -l /usr/bin
```

```
[
  {
    "filename": "apropos",
    "link_to": "whatis",
    "flags": "lrwxrwxrwx.",
    "links": 1,
    "owner": "root",
    "group": "root",
    "size": 6,
    "date": "Aug 15 10:53"
  },
  {
    "filename": "ar",
    "flags": "-rwxr-xr-x.",
    "links": 1,
    "owner": "root",
    "group": "root",
    "size": 62744,
    "date": "Aug 8 16:14"
  },
  {
    "filename": "arch",
    "flags": "-rwxr-xr-x.",
    "links": 1,
    "owner": "root",
    "group": "root",
    "size": 33080,
    "date": "Aug 19 23:25"
  }
]
```

netstat

```
netstat -apee | jc -p --netstat          # or:  jc -p netstat -apee
```

```
[
  {
    "proto": "tcp",
    "recv_q": 0,
    "send_q": 0,
    "local_address": "localhost",
    "foreign_address": "0.0.0.0",
    "state": "LISTEN",
    "user": "systemd-resolve",
    "inode": 26958,
    "program_name": "systemd-resolve",
    "kind": "network",
    "pid": 887,
    "local_port": "domain",
    "foreign_port": "*",
    "transport_protocol": "tcp",
    "network_protocol": "ipv4"
  },
  {
    "proto": "tcp6",
    "recv_q": 0,
    "send_q": 0,
    "local_address": "[::]",
    "foreign_address": "[::]",
    "state": "LISTEN",
    "user": "root",
    "inode": 30510,
    "program_name": "sshd",
    "kind": "network",
    "pid": 1186,
    "local_port": "ssh",
    "foreign_port": "*",
    "transport_protocol": "tcp",
    "network_protocol": "ipv6"
  },
  {
    "proto": "udp",
    "recv_q": 0,
    "send_q": 0,
    "local_address": "localhost",
    "foreign_address": "0.0.0.0",
    "state": null,
    "user": "systemd-resolve",
    "inode": 26957,
    "program_name": "systemd-resolve",
    "kind": "network",
    "pid": 887,
    "local_port": "domain",
    "foreign_port": "*",
    "transport_protocol": "udp",
    "network_protocol": "ipv4"
  },
  {
    "proto": "raw6",
    "recv_q": 0,
    "send_q": 0,
    "local_address": "[::]",
    "foreign_address": "[::]",
    "state": "7",
    "user": "systemd-network",
    "inode": 27001,
    "program_name": "systemd-network",
    "kind": "network",
    "pid": 867,
    "local_port": "ipv6-icmp",
    "foreign_port": "*",
    "transport_protocol": null,
    "network_protocol": "ipv6"
  },
  {
    "proto": "unix",
    "refcnt": 2,
    "flags": null,
    "type": "DGRAM",
    "state": null,
    "inode": 33322,
    "program_name": "systemd",
    "path": "/run/user/1000/systemd/notify",
    "kind": "socket",
    "pid": 1607
  }
]
```

/etc/passwd file

```
cat /etc/passwd | jc -p --passwd
```

```
[
  {
    "username": "root",
    "password": "*",
    "uid": 0,
    "gid": 0,
    "comment": "System Administrator",
    "home": "/var/root",
    "shell": "/bin/sh"
  },
  {
    "username": "daemon",
    "password": "*",
    "uid": 1,
    "gid": 1,
    "comment": "System Services",
    "home": "/var/root",
    "shell": "/usr/bin/false"
  }
]
```

ping

```
ping 8.8.8.8 -c 3 | jc -p --ping          # or:  jc -p ping 8.8.8.8 -c 3
```

```
{
  "destination_ip": "8.8.8.8",
  "data_bytes": 56,
  "pattern": null,
  "destination": "8.8.8.8",
  "packets_transmitted": 3,
  "packets_received": 3,
  "packet_loss_percent": 0.0,
  "duplicates": 0,
  "time_ms": 2005.0,
  "round_trip_ms_min": 23.835,
  "round_trip_ms_avg": 30.46,
  "round_trip_ms_max": 34.838,
  "round_trip_ms_stddev": 4.766,
  "responses": [
    {
      "type": "reply",
      "timestamp": null,
      "bytes": 64,
      "response_ip": "8.8.8.8",
      "icmp_seq": 1,
      "ttl": 118,
      "time_ms": 23.8,
      "duplicate": false
    },
    {
      "type": "reply",
      "timestamp": null,
      "bytes": 64,
      "response_ip": "8.8.8.8",
      "icmp_seq": 2,
      "ttl": 118,
      "time_ms": 34.8,
      "duplicate": false
    },
    {
      "type": "reply",
      "timestamp": null,
      "bytes": 64,
      "response_ip": "8.8.8.8",
      "icmp_seq": 3,
      "ttl": 118,
      "time_ms": 32.7,
      "duplicate": false
    }
  ]
}
```

ps

```
ps axu | jc -p --ps          # or:  jc -p ps axu
```

```
[
  {
    "user": "root",
    "pid": 1,
    "cpu_percent": 0.0,
    "mem_percent": 0.1,
    "vsz": 128072,
    "rss": 6784,
    "tty": null,
    "stat": "Ss",
    "start": "Nov09",
    "time": "0:08",
    "command": "/usr/lib/systemd/systemd --switched-root --system --deseria..."
  },
  {
    "user": "root",
    "pid": 2,
    "cpu_percent": 0.0,
    "mem_percent": 0.0,
    "vsz": 0,
    "rss": 0,
    "tty": null,
    "stat": "S",
    "start": "Nov09",
    "time": "0:00",
    "command": "[kthreadd]"
  },
  {
    "user": "root",
    "pid": 4,
    "cpu_percent": 0.0,
    "mem_percent": 0.0,
    "vsz": 0,
    "rss": 0,
    "tty": null,
    "stat": "S<",
    "start": "Nov09",
    "time": "0:00",
    "command": "[kworker/0:0H]"
  }
]
```

traceroute

```
traceroute -m 2 8.8.8.8 | jc -p --traceroute
# or:  jc -p traceroute -m 2 8.8.8.8
```

```
{
  "destination_ip": "8.8.8.8",
  "destination_name": "8.8.8.8",
  "hops": [
    {
      "hop": 1,
      "probes": [
        {
          "annotation": null,
          "asn": null,
          "ip": "192.168.1.254",
          "name": "dsldevice.local.net",
          "rtt": 6.616
        },
        {
          "annotation": null,
          "asn": null,
          "ip": "192.168.1.254",
          "name": "dsldevice.local.net",
          "rtt": 6.413
        },
        {
          "annotation": null,
          "asn": null,
          "ip": "192.168.1.254",
          "name": "dsldevice.local.net",
          "rtt": 6.308
        }
      ]
    },
    {
      "hop": 2,
      "probes": [
        {
          "annotation": null,
          "asn": null,
          "ip": "76.220.24.1",
          "name": "76-220-24-1.lightspeed.sntcca.sbcglobal.net",
          "rtt": 29.367
        },
        {
          "annotation": null,
          "asn": null,
          "ip": "76.220.24.1",
          "name": "76-220-24-1.lightspeed.sntcca.sbcglobal.net",
          "rtt": 40.197
        },
        {
          "annotation": null,
          "asn": null,
          "ip": "76.220.24.1",
          "name": "76-220-24-1.lightspeed.sntcca.sbcglobal.net",
          "rtt": 29.162
        }
      ]
    }
  ]
}
```

uptime

```
uptime | jc -p --uptime          # or:  jc -p uptime
```

```
{
  "time": "11:35",
  "uptime": "3 days, 4:03",
  "users": 5,
  "load_1m": 1.88,
  "load_5m": 2.0,
  "load_15m": 1.94,
  "time_hour": 11,
  "time_minute": 35,
  "time_second": null,
  "uptime_days": 3,
  "uptime_hours": 4,
  "uptime_minutes": 3,
  "uptime_total_seconds": 273780
}
```

XML files

```
cat cd_catalog.xml
```

```
<?xml version="1.0" encoding="UTF-8"?>
<CATALOG>
  <CD>
    <TITLE>Empire Burlesque</TITLE>
    <ARTIST>Bob Dylan</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Columbia</COMPANY>
    <PRICE>10.90</PRICE>
    <YEAR>1985</YEAR>
  </CD>
  <CD>
    <TITLE>Hide your heart</TITLE>
    <ARTIST>Bonnie Tyler</ARTIST>
    <COUNTRY>UK</COUNTRY>
    <COMPANY>CBS Records</COMPANY>
    <PRICE>9.90</PRICE>
    <YEAR>1988</YEAR>
  </CD>
  ...
```

```
cat cd_catalog.xml | jc -p --xml
```

```
{
  "CATALOG": {
    "CD": [
      {
        "TITLE": "Empire Burlesque",
        "ARTIST": "Bob Dylan",
        "COUNTRY": "USA",
        "COMPANY": "Columbia",
        "PRICE": "10.90",
        "YEAR": "1985"
      },
      {
        "TITLE": "Hide your heart",
        "ARTIST": "Bonnie Tyler",
        "COUNTRY": "UK",
        "COMPANY": "CBS Records",
        "PRICE": "9.90",
        "YEAR": "1988"
      }
    ]
  }
}
```

YAML files

```
cat istio.yaml
```

```
apiVersion: "authentication.istio.io/v1alpha1"
kind: "Policy"
metadata:
  name: "default"
  namespace: "default"
spec:
  peers:
  - mtls: {}
---
apiVersion: "networking.istio.io/v1alpha3"
kind: "DestinationRule"
metadata:
  name: "default"
  namespace: "default"
spec:
  host: "*.default.svc.cluster.local"
  trafficPolicy:
    tls:
      mode: ISTIO_MUTUAL
```

```
cat istio.yaml | jc -p --yaml
```

```
[
  {
    "apiVersion": "authentication.istio.io/v1alpha1",
    "kind": "Policy",
    "metadata": {
      "name": "default",
      "namespace": "default"
    },
    "spec": {
      "peers": [
        {
          "mtls": {}
        }
      ]
    }
  },
  {
    "apiVersion": "networking.istio.io/v1alpha3",
    "kind": "DestinationRule",
    "metadata": {
      "name": "default",
      "namespace": "default"
    },
    "spec": {
      "host": "*.default.svc.cluster.local",
      "trafficPolicy": {
        "tls": {
          "mode": "ISTIO_MUTUAL"
        }
      }
      }
    }
  ]
```