gron Greppable JSON

CONCEPTS

gron turns every leaf of a JSON document into one line such as json.user.tags[0] = "a"; and the root is always called json. grep keeps whole paths, so you see where a value lives. Do not rely on grep and sed pipelines in scripts; use jq there.

JSON file, URL or stdin
↓ gron
Assignments json.a.b[0] = 1;
↓ grep / sed / diff
Fewer lines still valid statements
↓ gron -u
JSON again
Round trip: flatten, filter, rebuild

READING INPUT

gron *<file>*
Flatten a JSON file

gron *<url>*
Fetch a URL and flatten the body

curl -s *<url>* **| gron**
Flatten JSON from stdin

gron -
Read from stdin explicitly

gron -s *<file>*
Treat each line as its own JSON object

READING INPUT (cont.)

gron -k *<url>*
Skip TLS certificate checks for a URL

gron -x *<proxy>* *<url>*
Fetch a URL through a proxy

gron --noproxy *<hosts>* *<url>*
Hosts that bypass the proxy, comma-separated

OUTPUT CONTROL

gron -m *<file>*
Monochrome, no ANSI colours

gron -c *<file>* **| less -R**
Force colour through a pager

gron --no-sort *<file>*
Keep document order, faster

gron -j *<file>*
Output as a JSON stream of [path,value]

gron --version
Print the version

SEARCHING

gron *<file>* **| grep** *<text>*
Show paths whose line matches

gron *<file>* **| grep -i** *<text>*
Case-insensitive match

SEARCHING (cont.)

gron *<file>* **| grep -v** *<text>*
Drop lines that match

gron *<file>* **| grep** **'^json\.***<key>***'**
Anchor to a top-level key

gron *<file>* **| grep -F '[0]'**
Match a literal array index

gron *<file>* **| grep -E** **'***<a>***|***<b>***'**
Match either of two patterns

gron *<file>* **| grep -c** **'***<text>***'**
Count matching lines

gron *<file>* **| grep '= "'**
Only string leaves

gron *<file>* **| grep -v '=** **\[\];\{};'**
Hide empty arrays and objects

REBUILDING JSON

gron -u *<file>*
Turn assignments back into JSON

gron *<file>* **| grep** *<text>* **| gron -u**
Keep a slice of a document as valid JSON

gron --ungron
Long form of -u

REBUILDING JSON (cont.)

gron -j *<file>* **| grep** *<text>* **| gron -j -u**
Round trip using the JSON stream form

gron *<file>* **| gron -v**
Print the values of the statements

gron *<file>* **| sed 's/^json\./json.***<new>***./' | gron -u**
Re-root the output under a new key

gron *<file>* **| sed -E 's/***<a>***\.//' | gron -u**
Flatten by stripping path segments

COMPARING AND EDITING

diff <(gron *<a>***) <(gron** *<b>***)**
See which paths differ between two files

diff -u <(gron *<a>***) <(gron** *<b>***)**
Unified diff of two documents

gron *<file>* **| sed 's/***<old>***/***<new>***/' | gron -u**
Substitute a value, then rebuild

gron *<file>* **| grep -v** *<path>* **| gron -u**
Delete a key

COMPARING AND EDITING (cont.)

gron *<file>* **| wc -l**
Count leaf values

EXIT CODES

Exit codes

Code	Meaning
0	OK
1	Failed to open file
2	Failed to read input
3	Failed to form statements
4	Failed to fetch URL
5	Failed to parse statements
6	Failed to encode JSON

RECIPES

Find a value in an unknown API

```
curl -s <url> | gron | grep -i <text>
# path found; now query it with jq
curl -s <url> | jq '.items[0].name'
```

Extract a sub-document

```
gron <file> | grep '^json.user' | gron -u > user.json
```

Diff two API responses

```
diff <(curl -s <url1> | gron) <(curl -s <url2> | gron)
```

Search JSON Lines logs

```
gron -s <log>.jsonl | grep error
# every line is numbered json[n]
```