yq YAML processor

START HERE

`yq '.a.b' <file>`
Read a value

`yq -i '.a.b = "x"' <file>`
Update in place

`yq '.[] | select(.n == "x")'`
Filter a list

`yq 'del(.a)' <file>`
Delete a key

`yq ea '. as $i ireduce({}; . * $i)' *.yml`
Merge files

`yq -o=json <file>`
YAML to JSON

`yq -p=json -o=yaml <file>`
JSON to YAML

`yq -n '.a = 1'`
Build from scratch

Input YAML, JSON, CSV... → `-p parse` → **Expression** .a[] | select(...) → `each match` → **Output** YAML by default → `-o encode` → **File or stdout** -i writes back

Expressions run once per document; eval-all sees every document of every file at once

CONCEPTS

This sheet is for mikefarah/yq v4 (the Go one). The Python yq wrapper around jq uses different flags and syntax.

Always single-quote the expression so the shell leaves $, |, * and parentheses alone.

READING

`yq '.<key>' <file>`
Print one value

`yq '.<a>.<b>[0]' <file>`
Walk keys and array index

`yq '.[]' <file>`
Every child of a list or map

`yq '.<key>[]' <file>`
Every item of a list under a key

`yq '."<odd key>"' <file>`
Key with dots or spaces

`yq '.<key> // "<default>"' <file>`
Fallback when null or false

`yq '.. | select(has("<key>"))' <file>`
Find nodes anywhere in the tree

`yq '.<key> | length' <file>`
Count items or characters

READING (cont.)

`yq 'keys' <file>`
List keys of a map

`yq '.<key> | tag' <file>`
Type of a node (!!str, !!int ...)

`yq <file>`
Pretty-print a whole file

UPDATING

`yq -i '.<key> = "<value>"' <file>`
Set a value in place
Without -i the result goes to stdout and the file is untouched

`yq '.<key> = "<value>"' <file>`
Preview the change on stdout

`yq -i '.<a>.<b>.<c> = 1' <file>`
Creates missing parents

`yq -i '.<n> |= . + 1' <file>`
Update using the old value

`yq -i '.<list> += ["<x>"]' <file>`
Append to a list

`yq -i '(.<a>, .<b>) = "<v>"' <file>`
Set several paths at once

`yq -i 'del(.<key>)' <file>`
Delete a key

UPDATING (cont.)

`yq -i 'del(.<list>[1])' <file>`
Delete an array element

`yq -i 'del(.<list>[] | select(. == "<v>"))' <file>`
Delete matching items

`yq -i 'with(.<path>; .a = 1 | .b = 2)' <file>`
Several edits under one path

SELECTING AND LISTS

`yq '.[] | select(.<k> == "<v>")' <file>`
Items where a field matches

`yq '.[] | select(.<k> == "x*")' <file>`
Glob match on a string
*The wildcards * match inside == comparisons on strings*

`yq '.[] | select(.<k> | test("<re>"))' <file>`
Regex match

`yq '(.[] | select(.<k> == "<v>") | .<f>) = "<new>"' <file>`
Update only matching items
Keep the whole left side in parentheses, or you get the filtered subset back

`yq 'map(.<k>)' <file>`
One field from every item

SELECTING AND LISTS (cont.)

`yq 'sort_by(.<k>)' <file>`
Sort a list by a field

`yq 'sort_by(.<k>) | reverse' <file>`
Sort descending

`yq 'unique' <file>`
Drop duplicate scalars

`yq 'unique_by(.<k>)' <file>`
Drop duplicates by field

FILES AND MERGING

`yq ea '<expr>' <file>...`
Evaluate over all documents together
ea is short for eval-all; it loads everything into memory

`yq ea '. as $i ireduce({}; . * $i)' <f1> <f2>`
Deep merge files, later wins

`yq ea 'select(fi == 0) * select(fi == 1)' <f1> <f2>`
Merge two files explicitly

`yq -i ea 'select(fi == 0) * select(fi == 1)' <f1> <f2>`
Merge f2 into f1 in place

`yq -n 'load("<f1>") * load("<f2>")'`
Merge via load, no input

FILES AND MERGING (cont.)

`yq '.a *+ .b' <file>`
Merge, appending arrays

`yq 'select(di == 1)' <file>`
Pick one document of a stream

`yq -N '<expr>' <file>`
Drop the --- separators

`yq -n '.a = 1'`
Create a document from nothing

`yq - < <file>`
Read from stdin

FORMATS

`yq -o=json <file>`
Output as JSON

`yq -p=json -o=yaml <file>`
JSON in, YAML out

`yq -o=json -I=0 <file>`
Compact one-line JSON

`yq -P <file>`
Normalise to pretty block style

`yq -o=props <file>`
Output Java properties

`yq -o=csv <file>`
List of lists to CSV

`yq -p=csv -o=yaml <file>`
CSV in as list of maps

FORMATS (cont.)

`yq -o=json -r '.<key>' <file>`
Raw string, no JSON quotes
For YAML output -r is already the default

`yq -e '.<key>' <file>`
Non-zero exit if null or false

VARIABLES, STRINGS AND ANCHORS

`NAME=<v> yq -i '.a = strenv(NAME)' <file>`
Inject an env var as string

`NAME=<v> yq -i '.a = env(NAME)' <file>`
Inject an env var, parsed as YAML
env(N)=12 becomes a number, strenv(N) stays "12"

`yq '(.. | select(tag == "!!str")) |= envsubst' <file>`
Expand ${VAR} in all strings

`yq '.<k> |= sub("<re>", "<new>")' <file>`
Regex replace in a string

SETUP AND HELP

`yq --version`
Show installed version

`yq --help`
Flags and commands