qkw

Generalized aliasing to simplify navigation, and executing complex multi-line terminal command sequences

---

Table of Contents

Design Philosophy :

The core idea behind building this application is to generalize the process of aliasing, so that anything could be *tagged, retrieved, searched, executed* and *annotated*.

Complex workflows are not straight forward to iterate quickly; be it for development, or automating a lengthy sequence of commands and it is often extremely hard to keep track of the commands, their options, variations, and redoing all the steps in a new environment.

qkw is developed to bridge this gap. Another major design parameter is portability i.e. the possibility to use the same labels in new environments where the targeted functionality is similar.

All the data is stored in two/three column tables, C:(L,V) or D:(L,V,E) databases, referred to in this documentation as "cmd", and "dir" respectively.

dir : {alias, directory}
cmd: {alias, command, note/explanation}

Serverless "sqlite database" is a core component of this application.

Installation :

- *The core application dependency libraries are **sqlite, fmt, yaml-cpp**, and compilation depends on **g++,cmake,make,pkg-config**.*
- *To keep the installation easy, a python based **buildscript.py** is available. **python3** binary is required, and no additional python related libraries are necessary.*
- *Open the **log file** generated by **buildscript.py** in the same directory and check for the issues.*
- *If sqlite, fmt, and yaml developer headers are available, then compilation is straightforward.*
- *Just do a **make install** followed by **make clean***
- *Set **QKW_CONFIG** environment file to **/etc/qkw-config.yaml***
- *After installation is successful, **qkw -version*** : would show the latest installed version

*The following files should be available in **/etc/qkw-config.yaml, /etc/userdata.db.***

Configuration :

*The configuration file is stored in **/etc/qkw/qkw-config.yaml**, and is linked to the environment variable **QKW_CONFIG**, which is mandatory for **qkw** to work. Configuration file can be saved anywhere, the only constraint is to set the environment variable.*

*The configuration file has two tables, **cmd**, and **dir.** Two separate lists of tables **{insert**, and **search}**, and the corresponding databases. So, in total 4 databases could be put in the configuration file.*

*For search operations using the switch "**s**", the search tables, and their corresponding databases are accessed. But, for **inserts,** a different table, and database is accessed. The data in the insert tables and it's database gets modified in most of the operations. "Search databases never get modified.*

*The search field in the config file can have multiple tables, but insert field should have only one table each; one for **cmd** and one for **dir** databases. It is recommended to have insert table added to the list of search tables.*

{cmd,dir}{search} = T0,T1,T2….,Tk | many tables allowed
{cmd,dir}{insert} = T0 | only one table

*As workflows get complex, one would like to have different databases and tables. The simplest possible case is set when **qkw** is installed.*

Building blocks of *options*

The single character options are combined with a single hyphen either separately or aggregated to create options. And, their permutations are valid in most cases and retain the same intended meaning.

*There are about **75+ options** that could be generated from basic character level sets or other sets*

`l`	**b**	List		
`i`	**b**	Insert		
`g`	**b**	Get		
`s`	**b**	Search		**cmd (C)**, and **dir (D)** tables are separately used for storing the commands, and directory paths separately C : (L,V,E), D: (L,V)
`d`	**b**	Delete		
`c`	**b**	check/create/		
`m`	**b**	Modify		
`e`	**b**	Edit (regex based)		
`A`	**a**	All		V:value, L:label, E: note/explanation are the column headers
`f`	**b**	File		
`T`	**b**	Table		The options act on the column labels and the meanings apply by context. **a**: auxilary, **b**:base auxilary option is used with other base options. Eg : qkw -C -lrA
`A`	**a**	All		
`{L,V,E}`	**ba**	Columns		
`r`	**a**	Row ids/integer		

`qkw -C -lA`, is the same as
`qkw -C -A -l` or
`qkw -A -lC` or
`qkw -lCA`

In order to make semantically more meaningful combinations, use **table type (C or D)**, succeeded by **operational commands.**

Eg : **qkw -C -lA** : list everything in the C insert table, is easier to understand than **qkw -l -CA**

qkw -T -lA : list all tables in 4 databases (cmd,dir).(insert,search)
qkw -D -lrA : list all directories including row ids

Help : `qkw -h <optionset>`,

Eg: `qkw -h -lDA`, will get a string showing the options, and how to apply. Note that **-h** switch precedes any option string.

Some specific options

These options are used as a whole, and don't work like options constructed out of single characters.

`cfgfile`	Show the path to the configuration file, as pointed to by the environment variable **QKW_CONFIG**
`dbname`	Database names used by the application for both the cmd, and dir tables, only for the insert operations
`cleartbls`	Clear all the content in the tables.
`mergetbls`	Merge more than one table into a final output table
`gettemplate`	Generate the empty table file for inserting/modifying data in the

	cmd table
addpath	Insert the path of the current directory
tbltype	Show the type of the tables, ‘cmd’/‘dir’
t{C,D}	Sub-option for temporarily changing the tables when using with other options
cpyT{C,D}	Copy the tables into multiple tables
ltrim{+r}	Trim the directory paths, particularly useful if you would like to use the same tags in a different environment, say inside container applications *docker*
lappend{+r}	Append to the left

All the options are loosely coupled, i.e., they could be used together or separately or per operation, and permutations of the options are valid for the majority of the options, unless, there is an functional block that is mapped to a smaller set of the given options set.

In that case, just do `qkw -D -df`, as `qkw -D -d` exists, and is executed first

[<table> <operation{option-modifer}>]

qkw -D -s <R1:string> -sLC <R2:string> -C -cT X1,X2,X3

* Search in the all the columns for D table
* search in the L columns of the C table for the given regex string
* create three new C tables, named X1,X2,X3

Note that all the options highlighted in single color are separately executed but provided as a simple string. **-D -s** *exemplifies loosely coupled options*

Options : Common for cmd, and dir tables

For the cmd table : **`qkw -D <options>`**, and dir table : **`qkw -C <options>`**

Options	#	Input Format	Explanation
-lA	0		List all options in the table
-l	1	<label,label,….>	Set of labels to return the values
-lr		<r,r,….>	Set of rowids, which are integers that could be obtained by -lA
-lrA	0		Show rows along with the L,V columns separated by a colon.
-s (or) **-sA**	1	<regex>	Search for all L,V columns, as well as E columns for the given regex string
-sL	1	<regex>	Search only in the labels column by the given regex string
-sV	1	<regex>	Search only in the values column by the given regex string, displays both

-i	1	<label>:<path>	Colon separated insertion, no gaps allowed. If path has spaces, use quotes
-g	1	<label>	Given label, get value only from the value column, only one entry is returned
-gr	1	<r>	Given rowid, get value only from the value column, only one entry is returned
-eL	2	<regex> <string>	Edit all rows in a table given a regular expression
-d	1	<label,label,….>	Delete an entire row regardless of the table type, given the labels
-cT	1	<table,table,...>	Create table, given the type to be of C, D respectively
-dr	1	<r,r,….>	Delete an entire row regardless of the table type, given the row ids
-wAf	1	<filename>	Write all the data from the table to file
-wf	1	<label,label…,label>	Write the specific labels to file
-if	1	<filename>	Insert from file, if L already exists/doesn’t, throws an error
-mf	1	<filename>	Modify from file, will overwrite with new data, doesn’t insert anew.
-df	1	<filename>	Delete data from file, only L column data will be used. Note that the file format should match for cmd, and dir separately
-dT	1	<table,table,….>	Delete the tables. User would be prompted for each deletion.
-i	1	<label>:<path>	Insert the path string
-mL	1	<iL>:<fL>,<iL>:<fL>,….	Modify the labels in the table, given initial label, and final label. If a label already exists, an error is thrown
-mrL	1	<r>:<fL>,<r>:<fL>,….	Row id integers could also be used to modify the labels
-mV	1	<label>:<new-value>	An existing label is mapped to a new value
-mrV	1	<r>:<new-value>	An existing row integer is mapped to a new value
-lT	0		List tables given, the table switch
-cleartbls	1	<table,table,…,>	Empty the table of it’s contents

Options : dir tables only

-ltrim	2	<regex> <label,label,….>	Trims to the left of the directory, given label.
-ltrim+r	2	<regex> <r,r,….>	Trims to the left of the directory, given rowid
-lappend	1	<label,label,….>	Append to the left of the directory, given labels
-lappend+r	1	<label,label,….>	Append to the left of the directory, given rowid
-addpath	1	<label>	Current directory is saved with the provided label. Is a shortcut for using *-iD <label>:<path>*

-cA	0		Check which directory paths are accessible
-eV	1	<regex> <string>	Replaces a regex matched expression for the value column for the entire column.
-cpyTD	1	<initial-table> <final-table>	Copy an existing table with it's contents of type D with L,V into a new table

Options : cmd tables only

-lA+s	1	<regex>	List content from all search tables
-addpath	1	<label>	Current directory is saved with the provided label. Is a shortcut for using *-iD <label>:<path>*

Options : all others

-TA	0		List all tables
-gettemplate	1	<filename>	Get a template file which could be used to insert data into cmd tables. Use -mf option to modify the content.
-cleartbls	1	<table,table,table>	Clear all the tables of off their contents
-mergetbls	1	<T,T...> <final-tbl>	Merge a list of tables into a final table. The final table could be one of the input table
-cfgfile	0		Retrieves the path to the configuration file
-tbl	1	<table>	Modifier option to change the table name temporarily, but database needs to be the same. This option precede the operational commands
-db	1	<string>	Temporary database to use with the rest of the options. If the C, and D tables exist, the options produce the expected result. This option should precede the operational commands
-exec	1	<string>	[Warning] The passed string is executed in a subshell and the result printed out. Be wary of what commands are passed to it.

Note :

- **[Important] Ensure that no sensitive information is saved in the tables, and the databases storing the tables are saved in a secure directory with restricted permissions.**
- L columns are unique and can't be empty, i.e. multiple values for same label can't be stored in the same table.
- Double check before deleting the tables or databases, it can often contain information that could save you a lot of time.
- *All options have a single hyphen -*, unlike the standard practice.
- When *merging* tables make sure that, no duplicates are allowed.

- **qkw** doesn't store any user information in the application binaries during compilation or later. All data stored or otherwise is external to the application in sqlite databases. Compilation is handled by the specific applications GNU make in the respective environments
- **[Warning]** Be careful when using macro runfast with commands that delete content or any operation that can have adverse effect.

Getting Started

Ask for help, by placing the -h before the command sequence

- `qkw -h -iC`
- `qkw -h -gettemplate`

Add entries in C table, and D table. Repeat the same with new labels, and values.

- `qkw -iC a:/usr/local`
- `qkw -iD qkw.clone:"git clone http://www.github.com/repo/qkw"`

*Get the value part only based on the tag. This data could be used to execute complex commands. Learn about the macros **runfast**, and **cd2** under Advanced Uses which effectively solves the problem of navigation and execution with a single option.*

- `qkw -gC a`
- `qkw -gD qkw.clone`

Visit your favorite directory and execute. Repeat the same for a few more directories

- `qkw -addpath myfav`
- `qkw -C -lA`

List out if the entries already exist by listing all entries

- `qkw -C -lA`
- `qkw -D -lrA`

Check if entries are accessible

- `qkw -D -cA`

Modify the added entries, and redo the step above

- `qkw -C -mL a:A`
- `qkw -D -mL a:B`

Delete entries by label

- `qkw -C -d A`
- `qkw -D -d B`

Create a few tables, and add to the config file

- Create a few tables,

```
qkw -C -cT C1,C2,C3
qkw -D -cT D1,D2
```

- Identify the table types, mismatched table types return incorrect formatting when queried
  `qkw -lTA`
- Get the config file, `qkw -cfgfile`
- Open it with a text editor
- Feel free to add the tables under {cmd,dir}.{insert,search} and note that insert tables are into which data is often added. Search tables are maintained separately for isolating the use cases.
- Remove the tables
  ```
  qkw -C -dT C1,C2,C3
  qkw -D -dT D1,D2
  ```

Add commands from file (insert, modify)

- `qkw -gettemplate mycmds.data`
- Open *mycmds.dat* with your favorite editor
- Under *label* add **cmd.1**
- Under *value* add **ls -ltrh >> dummy**
- Under *expl* add **"writing curr directory contents to file"**
- Save the file and exit
- `qkw -C -if mycmds.data`
- Now, modify the contents in mycmds.dat, and run
  `qkw -C -mf mycmds.data`
- List the contents by `qkw -C -lA`
- Modify the label `qkw -C -m cmd.1:ls.1`

Using a custom database and a table, both options are independent and can be used separately.

- `qkw -db path/to/mydata.db -tbl mytbl -C -lA`
- `qkw -db path/to/mydata.db -D -lA`
- `qkw -tbl mytbl -D -lA`

-tbl can help change table on which the operations are carried out instead of the insert tables. The same applies to **-db**, but make sure that the operations are carried out only if the table names match with those in the config file, since the application takes this as new db, but tables from config file. Hence in most cases, **-db <database> -tbl <table>** switches may be used together.

Similar functionality, same alias, and different tables

It just amounts to changing the tables and executing the commands with same tags when the environments differ.

Case 1

Any file system has a set of functions that are identical. Eg: create, copying, moving, listing, and deleting. As an example, consider hadoop, vs linux file systems

Operation	**Linux**	**Hadoop**
List	ls <args>	hdfs dfs -ls <args>
Make directory	mkdir <args>	hdfs dfs -mkdir <args>
Move	mv <args>	`hdfs dfs -mv <args>`

As it could be noticed, "ls, mkdir" are the operations with added paraphernalia to distinguish between the two systems. Not surprisingly the names are identical and modular. Now, saving the

commands in separate tables, one as **hadoop**, and other as **linux,** the user can tag the scripts with **runfast ls**, **runfast mkdir, etc** which becomes simpler without having to type a lot on the CLI.

Case 2
----
*Consider installing vim in yum-based vs apt-based systems. In such a scenario where commands are different, but functionality is the same, **qkw** comes in handy.*

Creating a temporary database using the switches **-db**, and **-tbl** database. NOTE: A dummy database is just a file. It could be created with the command **touch**

- Create a dummy database : `touch install.db`

- Create a new table, debian : `qkw -C -cT debian`
- Add a command to that table :
  `qkw -tbl debian -iC install-vi:”apt-get install -y vim”`
- List the commands : `qkw -tbl debian -C -lA`
- Create another new table centos : `qkw -cT centos`
- Add a command to that table :
  `qkw -tbl debian -iC install-vi:”yum install -y vim”`
- List the commands : `qkw -tbl centos -C -lA`

As you can see, **install-vi** remains same but commands are different. Note that, the insert database holds both the tables. If you prefer to write this to a new database. Just append `qkw -db myinstalls.db` and repeat the steps above.

- Create a dummy database : `touch install.db`

- `qkw -db install.db -tbl centos -C -iC install-vi:"yum install vim; yum install vim-enhanced"`
- `qkw -db install.db -tbl debian -C -iC install-vi:"apt-get install vim"`
- `qkw -db install.db -tbl debian -lCA`
- `qkw -db install.db -tbl centos -lCA`
- `qkw -db install.db -lTA`

-iC is a handy switch to add simple commands from the CLI using **<label>:<data>** without too many special characters which otherwise needs to be escaped. If there are *spaces in between the command*, put the data/value in quotes. The E field remains empty, which could be added later on using **-mf** switch

```
qkw -iC mvpdf:”mv /a/b/c/*.pdf /c/d/pdf-files/; wc -l”
```

For complex commands, *insertion via a template file is easier*.

Get template using `qkw -gettemplate <fileName>` and edit the file.

When you move to CentOS/Debian environment, open the config file, qkw-config.yaml, and store the path to the install.db under cmd, and insert, and start using the commands natively i.e. without using the temporary db switch.

Apply aliases recursively
The result here: two tables in the same custom database. You can even alias this custom database as follows

`qkw -iC install:"qkw -db install.db  "` # make sure to leave a gap to the right.

Now, using the *runfast macro,* and using the *install* as your tag/alias, you can perform operations on tables and databases as if it's a qkw database without having to put the db each time.

```
runfast install -tbl centos -C -lA
```

cDA or **cDrA** is a useful option to know *which directories are accessible* that are stored in your insert database.

```
[Y] : PA    /J/u/v      qkw -cDA
[Y] : U2    /J/a/b/c
[Y] : U3    /J/m/n/r/s
[N] : U4    /p1/q2/1
```

In the **Applications** section, check the macros to transform the tags executable macros

Advanced Uses

- *Moving to a new development environment:* modifying the directory paths, by using regular expressions

  Imagine a scenario where your computer is inaccessible and you are building a complex application. Assume you have dir tables that looks like the ones below. Especially the case if git repos are used.

By adding directory paths/ removing them, same tags/labels can be used as if it were the old environment.

After : lappend		**After : ltrim**		**Before : inaccessible computer**	
[N] : PA	/N/X/u/v	[N] : PA	u/v	[N] : PA	/J/u/v
[N] : U2	/N/X/a/b/c	[N] : U2	a/b/c	[N] : U2	/J/a/b/c
[N] : U3	/N/X/m/n/r/s	[N] : U3	m/n/r/s	[N] : U3	/J/m/n/r/s
[N] : U4	/p1/q2/1	[N] : U4	/p1/q2/1	[N] : U4	/p1/q2/1

Using the options **ltrm/lappend**, you could trim/append the relative part **/J for** several directories in a new environment.

```
qkw -cpyTC tJ tN # tJ:old name, tN: new name
qkw -tbl tN -ltrim ^/J ""
   or
qkw -tbl tN -lappend ^/J "/N/X"
qkw -D -cA # checks directory access
```

Note that, when new tables are created/added to the database, they are not automatically updated in the config file. It needs to be manually added. Using the command above, the list of all tables, their types and databases could be accessed.

```
qkw -dbname
qkw -cfgfile
qkw -lTA
```

- *Usage with docker, relative directory*

  Same as above. Most of modern development/deployments is based on containerized applications like docker. Using the approach above, the mounted file paths can be mapped to the same tags used for the host system, either by trimming or appending. Navigation and execution becomes trivial. However, qkw needs to be available with QKW_CONFIG set.

- *Using temporary databases, and tables therein which are not in the configuration file.*

  `qkw -db /path/to/another.db -tbl T -lCA`

  *Note that, **-db**, and **-tbl** switches precede the operational commands*

- *Template files for modifying/inserting commands*

  *T*his is the preferred approach if the commands have a lot of special characters like $,`,{, (etc.

  Eg: A command like this is easier to put via **-iCf** or **-mCf** operation rather than the **-iC label:value**

```
colorify() { n=$(bc <<< "$(echo ${1}|od -An -vtu1 -
w100000000|tr -d ' ')% 7"); echo -e "\e[3${n}m${1}\
e[0m"; }
```

- *Document or add help to command sequences :*

  The C tables have a note/explanation section where a short note could be added.
  Consider a bash sequence saved in a script as a function. In the cmd data file, add the following

myscript.sh

```
dirsize(){
    for dir in `ls -d */`; do
        du -sh $dir
    done
}
```

Content in expl section acts as a help string. Run `qkw -C -if forcmd.dat`
Visiting a directory, and doing :`_runfast dss`

forcmd.dat

```
label:
dss

value:
source path/to/myscript.sh;
dirsize

expl:
Check file size in the current
directory
---
```

Applications

qkw is designed to ensure ***portability*** i.e. the database file with all the commands is usually very small and can be emailed easily or saved to a ftp server.

Saving identical functional blocks with identical tags in different tables helps quickly iterate over.

```
qkw -tC cloudA -gC init
qkw -tC cloudB -gC init
```

'*init*' remains the same due to the functionality, but their paths, and commands at the backend

`qkw -tC home -gC init`	differ, and are stored in different tables. By pairing this with the Macro: “runfast”, it could be started with **`runfast init`**

To get the most out of **qkw**, source the following functions by placing them in a bashrc file or it’s equivalent for your Linux flavor. Options **gC, gD**, and **grC**, and **grC** are targeted for the purpose described below and could be used with other scripts as well.

Macro : “dir”

```
cd2(){
  cd `qkw -gD $1`
}
```

The macro sourced during startup in bashrc helps navigate to a any directory path aliased with a simple tag.

`source ~/.bashrc`

cd2 is just another bash function, perhaps faster than cd, and it could have any name.
gD retrieves just one V given a L and similarly **grD** based on the row id

```
cd2 src
cd2 note.todo
cd2 astro.png.sept02
cd2 astro.png.today
```

Behind the scenes, the paths could be very lengthy.

src : */this/is/a/crazy/long/long/long/loooong/far*
note.todo : /here/goes/another/dir/dir/dir/mynotes
astro.png.sept02 : /not/sure/where/it/is/dir/mynotes
astro.png.today : /all/my/organized/files/are/here/

Macro : “cmd”

[WARNING]

Avoid adding commands which delete data or those which potentially alter data irreversibly. An inadvertent application of `runfast <tag>` can have irreparable consequences.

Using this macro, the data in the **V or value** field is almost converted to a bash executable script. The functionality varies by the application that receives the data. Here it’s passed to “bash shell”, hence the text is interpreted as a bash script.

There are two ways :

Method-1

```
runfast(){
  qkw -gC $1 > /tmp/qkw-exec.tmpfile
  bash /tmp/qkw-exec.tmpfile
}
```

This macro helps executing an aliased complex command. Note that: $1 takes the alias from qkw. The builtin bash shell is used to execute the statements from a tmp file. For a different shell like Z, C, etc, please test before use.

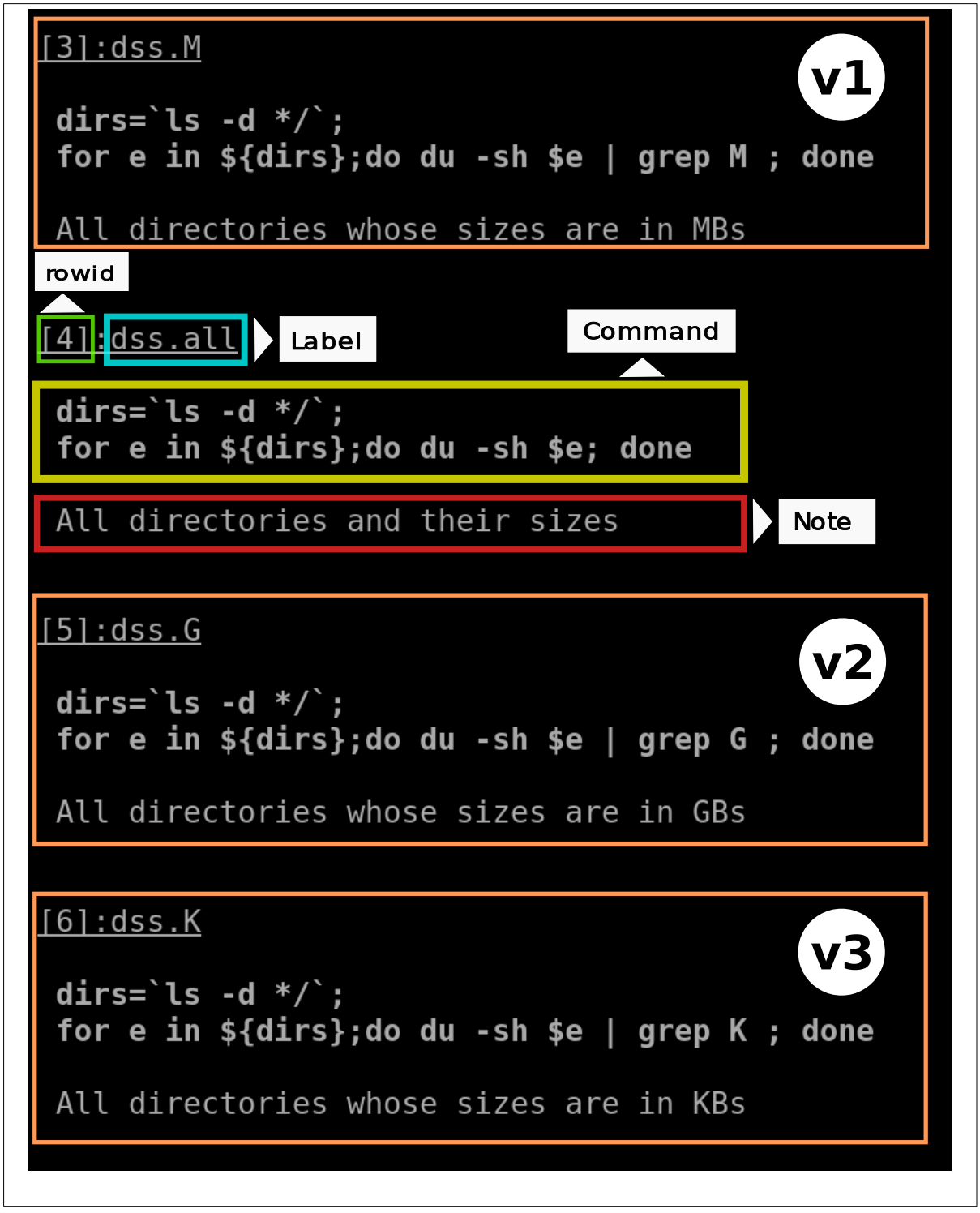
`source ~/.bashrc`

Method-2

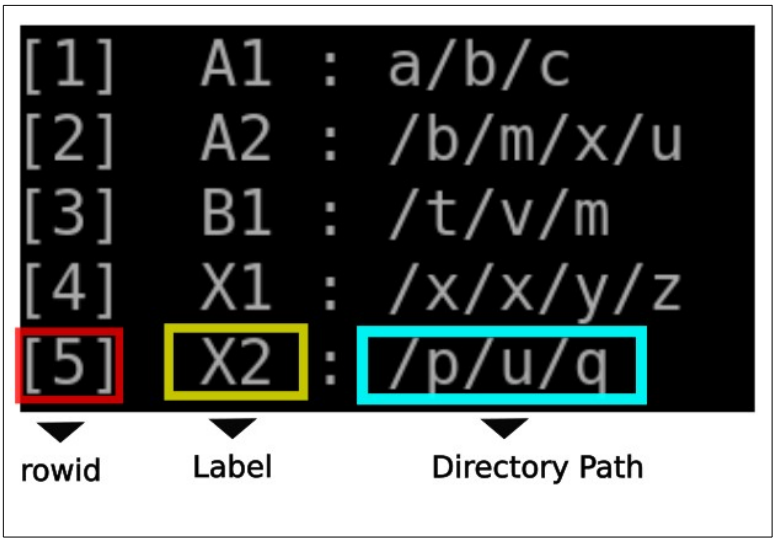
```
runfast(){
  str=`qkw -gC $1`
  qkw -exec "${str}"
}
```

This could also be used with the exec option which invokes a subprocess from the current calling process.

Sections of "*cmd*" tables : Three similar commands, and their variations



Sections of "*dir*" tables :



Examples:

- *Adding command versions with several options :*
  As an example, consider compiler options for C++; which are very long, not to mention the variations. The following exemplifies the application of both macros : **cd2**, and **runfast**

```
cd2 qkw.dev
runfast myapp.v1
```

Quickly go to the directory aliased by **qkw.dev**, and execute the command.

Retrieves the command below, and executes it

Behind the scenes, to retrieve the all the L,V, E fields, use `qkw -C -lA myapp.17,myapp.20`
Note that, the line breaks are preserved in command

```
[15]:myapp.20
 g++ -std=c++2a -g -o qkw  cmd.o dir.o utils.o qkw.o cli.o \
      `pkg-config --cflags --libs fmt` `pkg-config --cflags --libs sqlite3` \
      -L/usr/lib/x86_64-linux-gnu -lyaml-cpp -I/usr/include/yaml-cpp/

 Compiles for latest g++ std flag for a  minor commit from
 #717c2f9. Dated 05-12-2030

[16]:myapp.17
 g++ -std=c++17 -g -o qkw  cmd.o dir.o utils.o qkw.o cli.o \
      `pkg-config --cflags --libs fmt` `pkg-config --cflags --libs sqlite3` \
      -L/usr/lib/x86_64-linux-gnu -lyaml-cpp -I/usr/include/yaml-cpp/

 Compiles for latest g++17 std flag for a  minor commit from
 #717c2f9. Dated 05-12-2030
```

- *Save standard output/error along with the executable commands*: When development is in progress, a sequence of working commands are sought after. Using this tool, they

could be saved quickly with a note and a version. **E** column can be used to store the stdout as well via **expl.**

The data put in the **expl** section retains the structure. Hence the saved stdouts could be checked for *analysis*, *debugging*, *root-cause*, *etc*. The content at the very bottom is the stdout from running the command above in bold font.

```
[10]:dcl
 docker container ls


 REPOSITORY              TAG                 IMAGE ID            CREATED             SIZE
 aaaa1                   latest              e80095f5fed5        6 days ago          0.55GB
 bbbb1                   custom              12e08d927dea        8 days ago          0.81GB

 stdout as of Dec 35, 2030
```

- *Intermittent update of a file in a far-off directory* :

Case 1: Say, your multithreaded code is writing data to several directories, and need to copy their outputs into another directory for analysis. It helps, by just running `runfast cp.F`

```
[11]:cp.F

 cp -avr /a/b/c/F1 /m/n/F2 /x/y/z/F3 -t /p/q/r/s/t/v/
```

Case 2: Identifying changes in an application by changing configuration file data. Usually a sequence of operations like *start, stop, and reload* are to be repeated for any change, and could have multiple variations depending on the application of choice.

- `runfast cfg.1`, which opens the note for editing config file, and saving quickly without even having to visit the directory.
- `runfast rerun` will restart the application

Behind the scenes,

```
[9]:cfg.1

 vi /path/to/the/file/which/is/too/far/to/reach/gameapp.config
```

```
[14]:rerun

 systemctl myapp-server stop
 runfast ntw-reload
 myapp --update /path/to/the/file/which/is/too/far/to/reach/gameapp.config
 systemctl myapp-server start

 Update config file
```

- *Setting up labels to refer to minor/iterative improvements before finalizing*: Having labels named with an index that shows an *iteration* can be helpful instead of random tags.
  c.tue.1, c.tue.2
  The labels can always be renamed using `qkw -mL c.tue.20:c.final`