Bash+ Introduction

Bash+ is a set of functions written in Linux shell itself that allows users to access and use APIs of various providers from the command line. You can also say that bash+ is a method to command-line the API.

The goal of bash+ is to provide system admins/devops with an easy way to script operations vs different providers. Ideally, even a person with the most basic command of the Linux shell should be able to script operations.

Bash+ achieves its goal by:

1. Taking care of such tasks as managing authentication/authorization tokens and limit rates
2. Downloading and caching, when necessary, sites/accounts information
3. Providing built-in commands for most basic/common tasks
4. Providing tools that allow users to easily construct custom API requests by just copy-pasting relevant instructions from API manuals and without having to know the intricacies of accessing the given API

Bash+ can be seen as a revival of the original Unix/Linux philosophy of “Keep it simple - Keep it in a pipe”. That is, complex tasks should be carried out by a bunch of small simple commands organized in a pipe because there is beauty in simplicity.

Contrary to the common misconception, integrating APIs into the Linux shell is very easy. And bash+ already provides enough examples, conventions and common functions to make such an effort a breath. If every one of you writes a bash+ extension for an API you work with, the end result of our collective effort will be a super-shell that can do everything.

* Please note that the work on bash+ is ongoing and some aspects detailed in this guide may change

Summary:

Activating the Bash+ DynDNS extension

You start by sourcing files with bash+ functions. These files also include functions for DynDNS

```
[oskars@lab1 bash+]$
[oskars@lab1 bash+]$ ls *functions -1
common.functions
dnsme_api.functions
dnsme.functions
dnsme_sh.functions
dyndns.functions
dyndns_sh.functions
incap_api.functions
incap.functions
incap_help.functions
incap_sh.functions
[oskars@lab1 bash+]$
[oskars@lab1 bash+]$ while read i ; do source $i ; done < <(ls *functions -1)
[oskars@lab1 bash+]$
```

In case you wondered how big bash+ for DynDNS is, simple is not only beautiful and easy, it's also small

```
[oskars@lab1 bash+]$
[oskars@lab1 bash+]$ egrep '\S+' dyndns* | wc -l
309
[oskars@lab1 bash+]$
```

Once you have the functions in your shell and you run the first DynDNS command (any command), you will be prompted for credentials.
These are saved as variables inside your shell until you leave the current session

```
[oskars@lab1 bash+]$
[oskars@lab1 bash+]$ dyn
Customer: Largo
User: dynadmin
Password:
[oskars@lab1 bash+]$
```

Listing/Searching/Creating domains

You can list all domains in your account with the command *dyn ls*

Now lets create a new domain.
Notice the use of grep to filter domains. Bash+ is still bash. You are working with DynDNS in shell

```
[oskars@lab1 bash+]$
[oskars@lab1 bash+]$ dyn ls | grep oskar
[oskars@lab1 bash+]$
[oskars@lab1 bash+]$ dyn domain add oskar-test.com oskar@oskar-test.com 1800
{
  "function": "dyn_api",
  "msgs": [
    "create: New zone oskar-test.com created. Publish it to put it on our serv
    "setup: If you plan to provide your own secondary DNS for the zone, allow
6, 208.78.68.66, 2600:2001:0:1::66, 2600:2003:0:1::66"
  ],
  "data": {
    "zone_type": "Primary",
    "serial_style": "increment",
    "serial": 0,
    "zone": "oskar-test.com"
  }
}
{
  "function": "dyn_publish",
  "msgs": [
    "publish: oskar-test.com published"
  ]
}
[oskars@lab1 bash+]$
[oskars@lab1 bash+]$ dyn ls | grep oskar
oskar-test.com
[oskars@lab1 bash+]$
```

Creating records

Next, we populate the newly created domain with some records.

Creating a naked A record

```
[oskars@lab1 bash+]$
[oskars@lab1 bash+]$ dyn ls oskar-test.com
Id    Name    Type
[oskars@lab1 bash+]$
[oskars@lab1 bash+]$ dyn record add oskar-test.com @ A 1.1.1.1
{
  "function": "dyn_api",
  "msgs": [
    "add: Record added"
  ],
  "data": {
    "zone": "oskar-test.com",
    "ttl": 1800,
    "fqdn": "oskar-test.com",
    "record_type": "A",
    "rdata": {
      "address": "1.1.1.1"
    },
    "record_id": 0
  }
}
{
  "function": "dyn_publish",
  "msgs": [
    "publish: oskar-test.com published"
  ]
}
[oskars@lab1 bash+]$
[oskars@lab1 bash+]$ dyn ls oskar-test.com
Id           Name    Type
268010068    @       A
[oskars@lab1 bash+]$
```

Creating a cname

```
[oskars@lab1 bash+]$ dyn record add oskar-test.com yahoo CNAME yahoo.com
{
  "function": "dyn_api",
  "msgs": [
    "add_node: Reactivating zone node",
    "add: Record added"
  ],
  "data": {
    "zone": "oskar-test.com",
    "ttl": 1800,
    "fqdn": "yahoo.oskar-test.com",
    "record_type": "CNAME",
    "rdata": {
      "cname": "yahoo.com."
    },
    "record_id": 0
  }
}
{
  "function": "dyn_publish",
  "msgs": [
    "publish: oskar-test.com published"
  ]
}
[oskars@lab1 bash+]$
[oskars@lab1 bash+]$ dyn ls oskar-test.com
Id           Name     Type
268011299    yahoo    CNAME
268010068    @        A
[oskars@lab1 bash+]$
```

Created a naked text record and listing that record by its id to see all its values

```
[oskars@lab1 bash+]$ dyn record add oskar-test.com @ TXT "Everything is fine"
{
  "function": "dyn_api",
  "msgs": [
    "add: Record added"
  ],
  "data": {
    "zone": "oskar-test.com",
    "ttl": 1800,
    "fqdn": "oskar-test.com",
    "record_type": "TXT",
    "rdata": {
      "txtdata": "Everything is fine"
    },
    "record_id": 0
  }
}
{
  "function": "dyn_publish",
  "msgs": [
    "publish: oskar-test.com published"
  ]
}
[oskars@lab1 bash+]$
[oskars@lab1 bash+]$ dyn ls oskar-test.com
Id          Name     Type
268011299   yahoo    CNAME
268011635   @        TXT
268010068   @        A
[oskars@lab1 bash+]$
[oskars@lab1 bash+]$ dyn ls oskar-test.com 268011635
Id          Name    Type    TTL     Rdata    Target/Value
268011635   @       TXT     1800    -        Everything is fine
[oskars@lab1 bash+]$
```

Listing/filtering domain records

You have many options for listing records.
You can use *dyn ls <domain> all* to print all records
If you have many records, this operation can take a while to finish

```
[oskars@lab1 bash+]$
[oskars@lab1 bash+]$ dyn ls oskar-test.com all
Id          Name     Type     TTL     Rdata    Target/Value
268011299   yahoo    CNAME    1800    -        yahoo.com.
268011635   @        TXT      1800    -        Everything is fine
268010068   @        A        1800    -        1.1.1.1
[oskars@lab1 bash+]$
```

dyn ls is in fact a complex command that uses grep to filter records.
You should enclose the grep string in quotes when passing it to dyn ls
The examples below include:

1. Filtering record by name yahoo
2. Filtering record by type TXT
3. Filtering all naked A records
4. Filtering all naked records

```
[oskars@lab1 bash+]$
[oskars@lab1 bash+]$ dyn ls oskar-test.com yahoo
Id          Name     Type     TTL     Rdata    Target/Value
268011299   yahoo    CNAME    1800    -        yahoo.com.
[oskars@lab1 bash+]$
[oskars@lab1 bash+]$ dyn ls oskar-test.com TXT
Id          Name     Type     TTL     Rdata    Target/Value
268011635   @        TXT      1800    -        Everything is fine
[oskars@lab1 bash+]$
[oskars@lab1 bash+]$ dyn ls oskar-test.com '@\s+A'
Id          Name     Type     TTL     Rdata    Target/Value
268010068   @        A        1800    -        1.1.1.1
[oskars@lab1 bash+]$
[oskars@lab1 bash+]$ dyn ls oskar-test.com '\s@\s'
Id          Name     Type     TTL     Rdata    Target/Value
268011635   @        TXT      1800    -        Everything is fine
268010068   @        A        1800    -        1.1.1.1
[oskars@lab1 bash+]$
```

Deleting domains/records

To delete domain and records, use *dyn rm*
For example *dyn rm oskar-test.com* will delete the domain
This command asks for confirmation. In case you are running it in a script, use regular bash redirection to simulate user input

```
[oskars@lab1 bash+]$
[oskars@lab1 bash+]$ dyn ls oskar-test.com
Id    Name    Type
[oskars@lab1 bash+]$
[oskars@lab1 bash+]$
[oskars@lab1 bash+]$
[oskars@lab1 bash+]$ dyn rm  oskar-test.com
Press Y/y to delete domain oskar-test.com: ^C
[oskars@lab1 bash+]$
[oskars@lab1 bash+]$ dyn rm  oskar-test.com <<<y
{
  "function": "dyn_api",
  "msgs": [
    "remove: Zone removed"
  ],
  "data": {}
}
[oskars@lab1 bash+]$
[oskars@lab1 bash+]$ dyn ls oskar-test.com
{
  "function": "dyn_api",
  "msgs": [
    "zone: No such zone",
    "get_tree: No such zone in your account"
  ]
}
Id    Name    Type
[oskars@lab1 bash+]$
```

You delete records by their id. Multiple records can be specified at the same time. Confirmation is not needed

```
}
[oskars@lab1 bash+]$
[oskars@lab1 bash+]$ dyn ls oskar-test.com
Id           Name    Type
268014513    @       A
268014523    @       A
[oskars@lab1 bash+]$
[oskars@lab1 bash+]$ dyn rm  oskar-test.com 268014513 268014523
{
  "function": "dyn_api",
  "msgs": [
    "delete: Record will be deleted on zone publish"
  ],
  "data": {}
}
{
  "function": "dyn_api",
  "msgs": [
    "delete: Record will be deleted on zone publish"
  ],
  "data": {}
}
{
  "function": "dyn_publish",
  "msgs": [
    "publish: oskar-test.com published"
  ]
}
[oskars@lab1 bash+]$
[oskars@lab1 bash+]$ dyn ls oskar-test.com
Id    Name    Type
[oskars@lab1 bash+]$
[oskars@lab1 bash+]$
```

Updating existing records

You can also update existing records, using the same command *record add*

```
[oskars@lab1 bash+]$
[oskars@lab1 bash+]$
[oskars@lab1 bash+]$ dyn ls oskar-test.com all
Id          Name     Type    TTL     Rdata    Target/Value
268037363   test1    A       1800    -        1.1.1.1
[oskars@lab1 bash+]$
[oskars@lab1 bash+]$ dyn record add oskar-test.com test1 A 2.2.2.2
{
  "function": "dyn_api",
  "msgs": [
    "update: Record updated"
  ],
  "data": {
    "zone": "oskar-test.com",
    "ttl": 1800,
    "fqdn": "test1.oskar-test.com",
    "record_type": "A",
    "rdata": {
      "address": "2.2.2.2"
    },
    "record_id": 0
  }
}
{
  "function": "dyn_publish",
  "msgs": [
    "publish: oskar-test.com published"
  ]
}
[oskars@lab1 bash+]$
[oskars@lab1 bash+]$ dyn ls oskar-test.com all
Id          Name     Type    TTL     Rdata    Target/Value
268037388   test1    A       1800    -        2.2.2.2
[oskars@lab1 bash+]$
```

Custom API requests

The command “record add” above can process records of the type A, CNAME and TXT. It doesn’t work with MX or SRV records. However, you can take advantage of bash+ functions to easily create your own requests to the API.

Bash+ has function *dyn_api* to access API that can be also evoked as *dyn api*

```
[oskars@lab1 bash+]$ declare -f dyn_api
dyn_api ()
{
    dyn_token_get;
    local res exit_code;
    res=$( curl -Ss -H content-type:application/json -H Auth-Token:\ $dyn_token "$@" 2>&1);
    exit_code=$?;
    if [[ x$exit_code != x0 ]]; then
        echo Curl error mesg: $res 1>&2;
        echo Curl error code: $exit_code 1>&2;
        return $exit_code;
```

dyn_api takes care of some aspects such as initiating and maintaining the authorization token. You only need to complete your request with relevant arguments to curl

So lets take as an example creating an MX record
From the online API manual

REST Syntax

/REST/MXRecord/ POST — Create a new MX Record on the zone/node indicated.

HTTP Action — POST

URI — https://api.dynect.net/REST/MXRecord/<zone>/<fqdn>/

Arguments — Click for More Info

- hash `rdata` — **Required.** RData defining the record to add.
  - string `exchange` — **Required.** Hostname of the server responsible for accepting mail me
  - string `preference` — **Required.** Numeric value for priority usage. Lower value takes pre

    two records of the same type exist on the zone/node. Default = 10.
- string `ttl` — TTL for the record in seconds. Set to “0” to use zone default.

HTTP Action POST means that your request should include *-X POST* switch
dyn api -X POST

You provide arguments with switch *--data*
dyn api -X POST --data <arguments>

API expects arguments in the json format.
You can use bash+ *args2json* function to easily construct the json
For your MX record, *args2json* would look like this:
args2json rdata.preference=0 rdata.exchange=mail.oskar-test.com ttl=14400

And this is the json for API you receive from *args2json*:
{"rdata":{"preference":0,"exchange":"mail.oskar-test.com"},"ttl":14400}

In the screenshot below I first save the output of *args2json* to a variable and then use this variable in my command.

The last parameter to *dyn_api* is the API URI

In the screenshot below it all comes together as we create the MX record

```
[oskars@lab1 bash+]$
[oskars@lab1 bash+]$ dyn ls oskar-test.com MX
Id    Name    Type    TTL    Rdata    Target/Value
[oskars@lab1 bash+]$
[oskars@lab1 bash+]$ args2json rdata.preference=0 rdata.exchange=mail.oskar-test.com ttl=14400
{"rdata":{"preference":0,"exchange":"mail.oskar-test.com"},"ttl":14400}
[oskars@lab1 bash+]$
[oskars@lab1 bash+]$ rdata=$(args2json rdata.preference=0 rdata.exchange=mail.oskar-test.com ttl=1
[oskars@lab1 bash+]$
[oskars@lab1 bash+]$ dyn_api -X POST --data "$rdata" \
> https://api.dynect.net/REST/MXRecord/oskar-test.com/oskar-test.com
{
  "status": "success",
  "data": {
    "zone": "oskar-test.com",
    "ttl": 14400,
    "fqdn": "oskar-test.com",
    "record_type": "MX",
    "rdata": {
      "preference": 0,
      "exchange": "mail.oskar-test.com."
    },
    "record_id": 0
  },
  "job_id": 3554428914,
  "msgs": [
    {
      "INFO": "add: Record added",
      "SOURCE": "BLL",
      "ERR_CD": null,
      "LVL": "INFO"
    }
  ]
}
[oskars@lab1 bash+]$
[oskars@lab1 bash+]$ dyn publish changes oskar-test.com
"publish: oskar-test.com published"
[oskars@lab1 bash+]$
[oskars@lab1 bash+]$ dyn ls oskar-test.com MX
Id          Name    Type    TTL      Rdata          Target/Value
267756077   @       MX      14400    preference=0   mail.oskar-test.com.
[oskars@lab1 bash+]$
```

Appendix A. Notes for scripters

Bash+ takes care of several aspects of accessing DynDNS:

1) Token management
2) Publishing changes

Basically whenever you use bash+ dyndns extensions, an authorization token is automatically requested and kept alive. Bash+ functions don't delete the ticket after completing their requests. However, DynDNS considers it a good practice to close the ticket when the task is completed. In case you want to close the ticket, use *dyn token kill* command.

Sometimes you may want to run asynchronous requests to API with different tokens to avoid the rate limit (5 requests/second). In this case you would run your requests in subshells at the background. To cause bash+ to request a ticket for your subshell, unset dyn_token variable after you start the subshell. You can close the ticket with *dyn token kill* command at the end of the subshell. Here is an example from one of the bash+ functions

```
s@lab1 bash+]$
s@lab1 bash+]$ declare -f dyn_record_ls | s
  ( dyn debug start background subshell;
  unset dyn_token;
  dyn_api https://api.dynect.net/REST/${typ
  dyn token kill;
s@lab1 bash+]$
s@lab1 bash+]$
```

Another aspect to consider is the publishing of changes. Most API commands of DynDNS only submit their changes to the API without actually implementing them. You need to send a publish command to activate the new configuration. Keep in mind that if you delete the authorization token, all updates submitted using this token are lost. You need to publish your changes before you close the token

Bash+ has two commands *dyn publish zone <domain>* for newly created domains and *dyn publish changes <domain>* to publish new/updated records.

All commands that add/update domain/records that come with bash+ publish changes after every API request. In case you work with many records in the same domain, you may prefer to save on requests and first submit all your changes using *dyn_api* command and only then publish them.

The following error codes are returned by *dyn api*. You can use them to identify the source of error and also to know what kind of output is saved in res variable
0 - Request succeeded
1 - The API informed you that your request failed
2 - The request failed and the response is not in json format (DynDNS responds to certain errors with html pages)
10+ Curl failed and *res* contains the curl error message. All curl error codes are recalculated by adding 10

Please notice that the next version of the function *dyn api* will provide no output besides printing short error messages to stderr if the request failed. That is, you don't pipe or save to variables the output of your requests. If you have just run a request to API using *dyn api* command, its output is stored in the variable *res.* Errors are also saved into this variable in case you want to see the complete response of a failed request

```
[oskars@lab1 bash+]$
[oskars@lab1 bash+]$ echo $res | jq
{
  "status": "success",
  "data": {
    "zone": "oskar-test.com",
    "ttl": 1800,
    "fqdn": "test1.oskar-test.com",
    "record_type": "A",
    "rdata": {
      "address": "1.1.1.1"
    },
    "record_id": 0
  },
  "job_id": 3583744355,
  "msgs": [
    {
      "INFO": "add_node: Reactivating zone node",
      "SOURCE": "BLL",
      "ERR_CD": null,
      "LVL": "INFO"
    },
    {
      "INFO": "add: Record added",
      "SOURCE": "BLL",
      "ERR_CD": null,
      "LVL": "INFO"
    }
  ]
}
[oskars@lab1 bash+]$
```

So, if you want to view or process the response from the API in a more friendly format, you can pipe the variable res thru the function *json2bash* provided by bash+

```
}
[oskars@lab1 bash+]$
[oskars@lab1 bash+]$ echo $res |  json2bash
status                  = success
data zone               = oskar-test.com
data ttl                = 1800
data fqdn               = test1.oskar-test.com
data record_type        = A
data rdata address      = 1.1.1.1
data record_id          = 0
job_id                  = 3583744355
msgs 0 INFO             = add_node: Reactivating zone node
msgs 0 SOURCE           = BLL
msgs 0 ERR_CD           = null
msgs 0 LVL              = INFO
msgs 1 INFO             = add: Record added
msgs 1 SOURCE           = BLL
msgs 1 ERR_CD           = null
msgs 1 LVL              = INFO
[oskars@lab1 bash+]$
```

For your own functions, you can use dyn_msg command to print info/error messages
The convention in bash+ is to always send messages to stderr by using 1>&2 redirection

```
[oskars@lab1 bash+]$
[oskars@lab1 bash+]$ dyn api -X PUT --data $(args2json publish=true) https://api.dyn
[oskars@lab1 bash+]$
[oskars@lab1 bash+]$ echo $res | dyn_msg 1>&2
{
  "function": "dyn_api",
  "msgs": [
    "changeset: No changes to apply.",
    "publish: Could not publish oskar-test.com. No changes to apply."
  ]
}
[oskars@lab1 bash+]$
[oskars@lab1 bash+]$ declare -f dyn_msg
dyn_msg ()
{
    cat /dev/stdin | jq -e '{ function: "'${FUNCNAME[1]}'", msgs: [.msgs[].INFO] }'
}
[oskars@lab1 bash+]$
```