Bash+ Introduction

Bash+ is a set of functions written in Linux shell itself that integrates APIs of various providers into the command line of the shell. 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.

Summary:

Getting started with the bash+ Incapsula extension

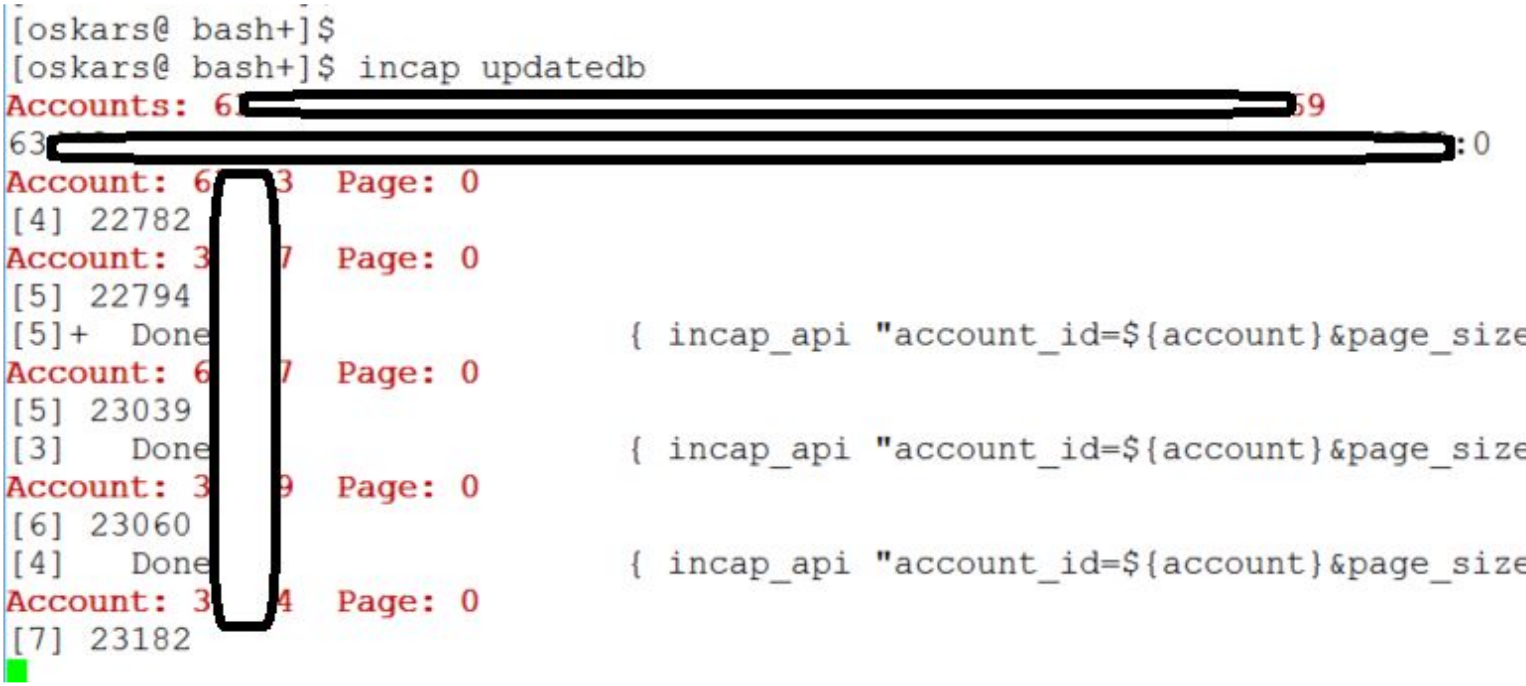
You start by sourcing files with bash+ functions. These files also include functions for Incapsula

```
[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 $i ; done < <(ls *functions -1)
[oskars@lab1 bash+]$
```

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

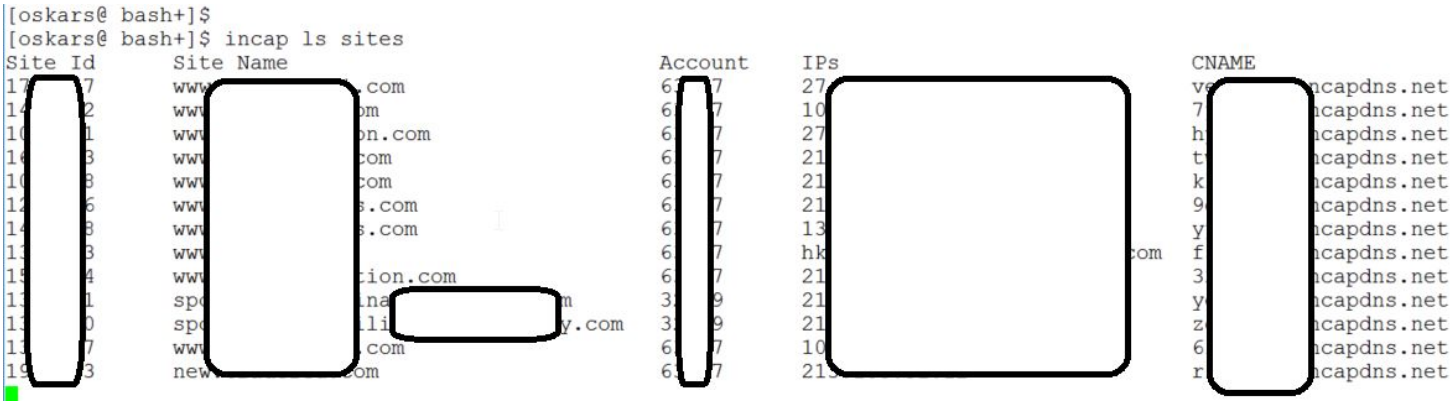
```
[oskars@ bash+]$
[oskars@ bash+]$ incap
API ID:
API Key:
[oskars@ bash+]$
```

Finally, you download your Incapsula configuration by running *incap updatedb*



incap updateb will be explained in more detail later in one of the next sections

Once the Incapsula configuration downloaded, you can list all your sites by running *incap ls sites*



You can filter the list using grep.

```
oskars@ bash+]$
oskars@ bash+]$ incap ls sites | grep oskar
oskars@ bash+]$
```

There is no site whose name or origin includes oskar at this point.

Next, lets create a new Incapsula site using command *incap site mk*

```
[oskars@ bash+]$
[oskars@ bash+]$ incap site mk help

<function> <account id> <site name> <origin ip>

[oskars@ bash+]$
[oskars@ bash+]$ incap site mk 3[redacted]9 demo.oskar-test.com 192.1.1.1
Creating site demo.oskar-test.com
"OK"
Setting ignore_ssl to read the incapdns record:
"OK"
[oskars@ bash+]$
[oskars@ bash+]$ incap ls sites | grep oskar
11652521     demo.oskar-test.com  3[redacted]9    192.1.1.1    yk35q6z.x.incapdns.net
[oskars@ bash+]$ _
```

This time when we filter *incap ls sites* with grep, the new site shows up

Introduction into bash+ for Incapsula by example

This is a real world case of the Incapsula extension of bash+ in action.
The request came to disable TCP-pre-pooling on all sites under a certain Incapsula account (Incapsula account is a group of sites).
So you go to Incapsula online API Doc Manual and find relevant instructions

Advanced Caching Settings

Use this operation to modify advanced caching settings.

```
/api/prov/v1/sites/performance/advanced
```

Parameters:

Name	Description	Optional
api_id	API authentication identifier	
api_key	API authentication identifier	
site_id	Numeric identifier of the site to operate on.	
param	Name of configuration parameter to set, see table below	
value	According to the param value, see table below	

Name	Description
async_validation	Sets Async validation. Pass "true" or "false" in the value parameter
tcp_pre_pooling	TCP Pre-Pooling, Pass "true" or "false" in the value parameter

Basically, to disable pre pooling you need to send a POST request to
https://my.incapsula.com/api/prov/v1/sites/performance/advanced

Your request should look like this:
api_id=<Your API id>&api_key=<Your API key>&site_id=<Your site id>&param=tcp_pre_pooling&value=false

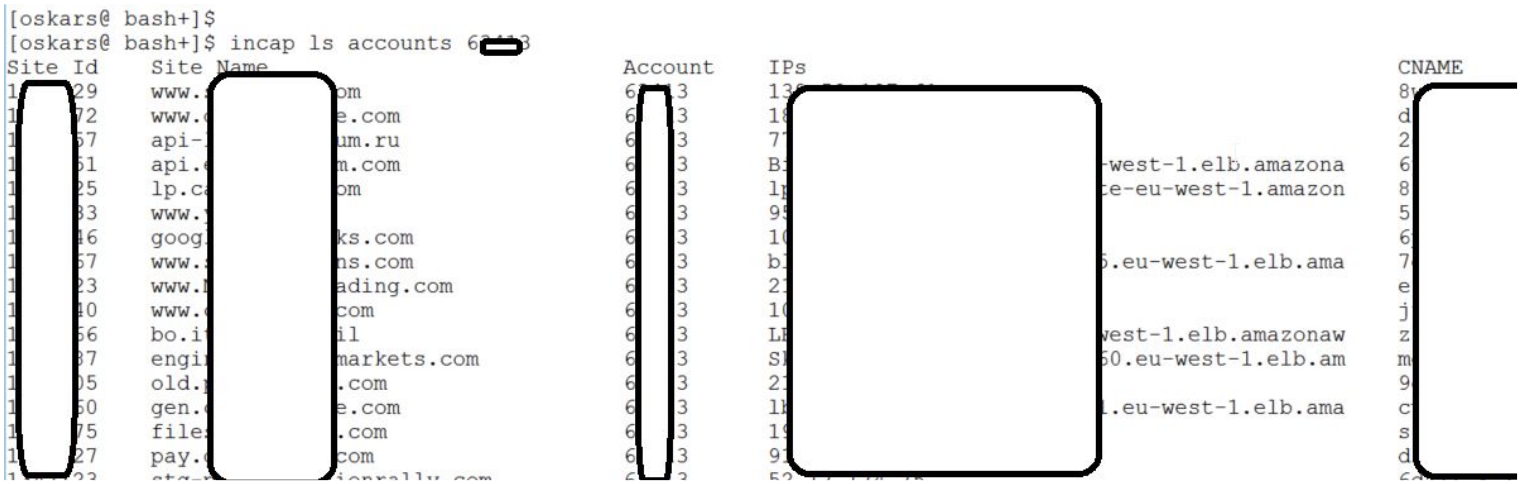
To run such a request you use command *incap api*. The command simplifies your task. You only need to copy relevant arguments from the API Manual. Your command would look like this:

incap api "site_id=<Your site id>&param=tcp_pre_pooling&value=false"
/api/prov/v1/sites/performance/advanced

The only parameter missing with this command is the site id. In the next page you will find the ids of all sites of that account

You can list your sites and their configuration using command *incap ls*.
For this example, you run *incap ls accounts* to receive the list of your accounts

Next, you run *incap ls accounts <account id>* and receive the list of all sites under that account



To save all site ids into the variable , you run:
ids=$(incap ls accounts <account id> | egrep -o ^[0-9]+)

And finally you run the entire command as:

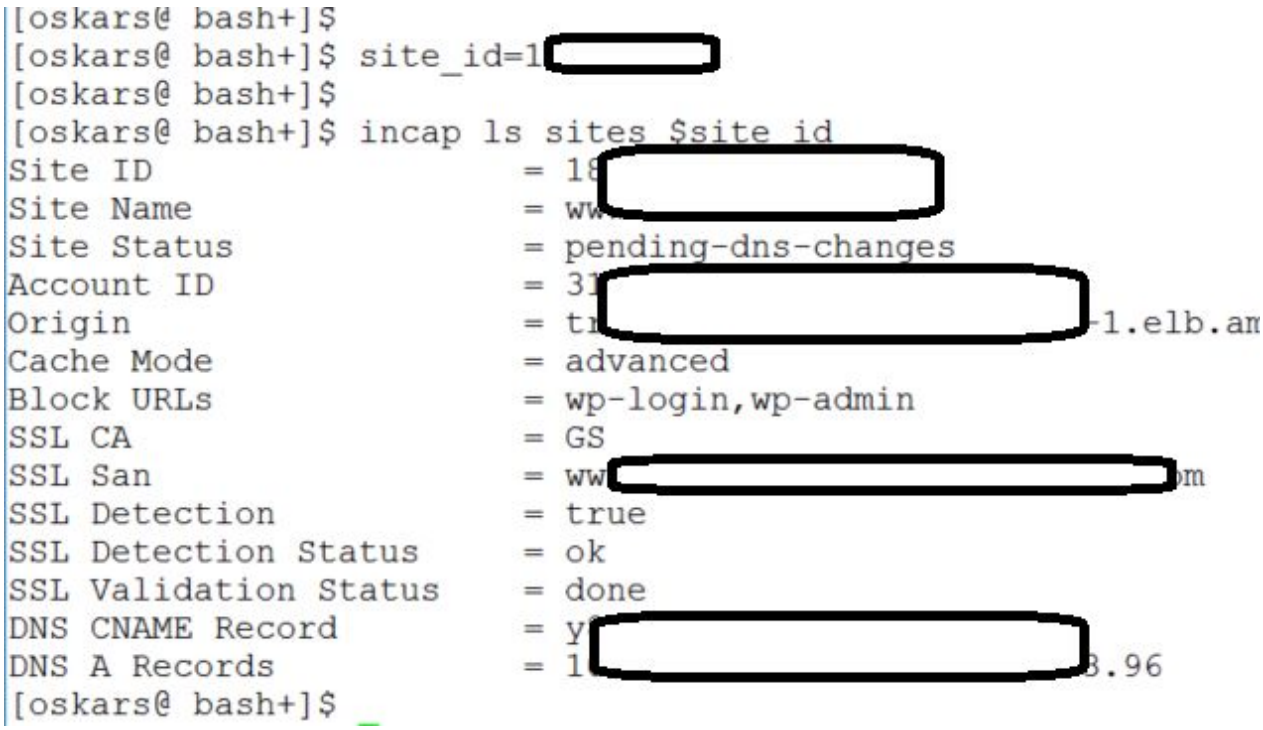
for id in $ids ; do
incap api "site_id=${id}&param=tcp_pre_pooling&value=false" /api/prov/v1/sites/performance/advanced
done

Listing Incapsula sites,accounts and configuration

At the top level bash+ for Incapsula displays 4 pseudo directories: accounts, sites, origins and cache

```
[oskars@ bash+]$
[oskars@ bash+]$ incap ls
accounts
sites
origins
cache
[oskars@ bash+]$
```

You have already seen how to work with sites and accounts by now.
One additional command is *incap ls sites <site id>* prints a summary of the site configuration



Origins work on the same principle as accounts. There is a list of all origins your sites point to.

```
[oskars@ bash+]$ incap ls origins
Origin
21
21
27
21
27
27
21
21
10
27
BE                                  714351734.eu-west-1.elb.amazonaws.com
27
P1                                  -746990009.eu-west-1.elb.amazonaws.com
21
21
WE                                  amazonaws.com
P1                                  8.eu-west-1.elb.amazonaws.com
LE                                  elb.amazonaws.com
Ap                                  elb.amazonaws.com
21
14
```

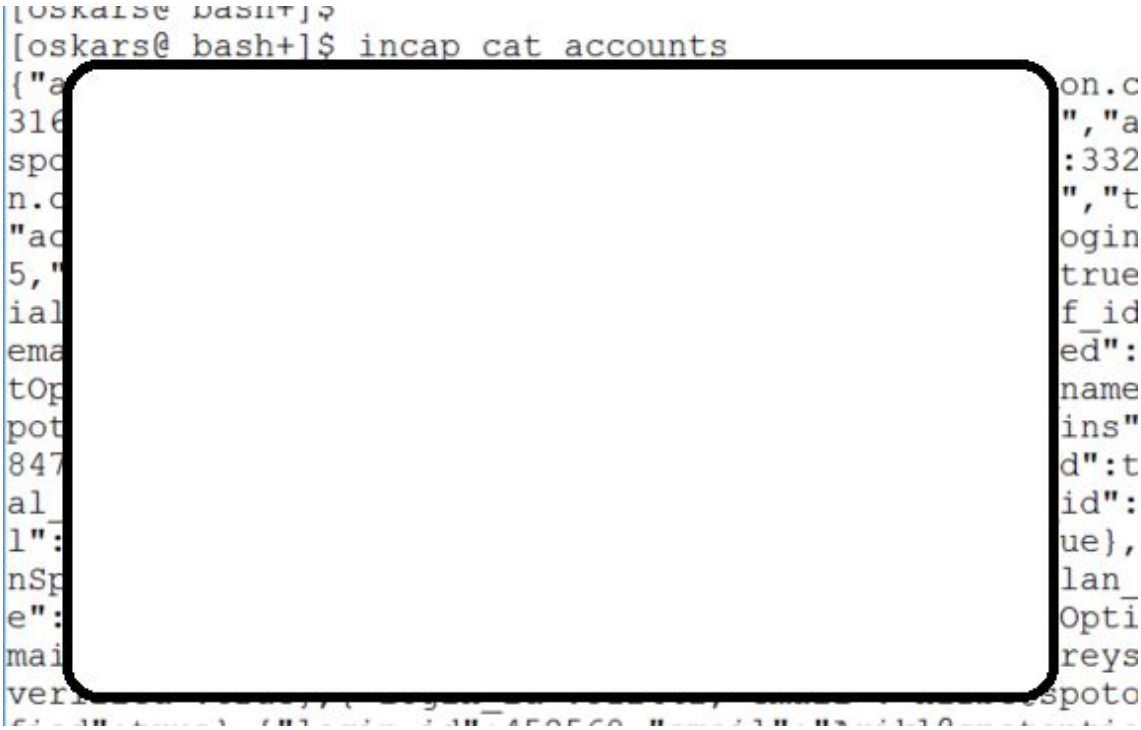
You can also print all sites that point to a certain IP or hostname.
This is useful when you need to get the ids of all sites that should be migrated from one data center to another

incap ls origins <origin>

incap ls origins is not only easier to work with. *incap ls sites/accounts* truncates long hostnames to keep the output in a table. *incap ls origins <origin ip/hostname>* will output exactly the sites you need with no place for errors

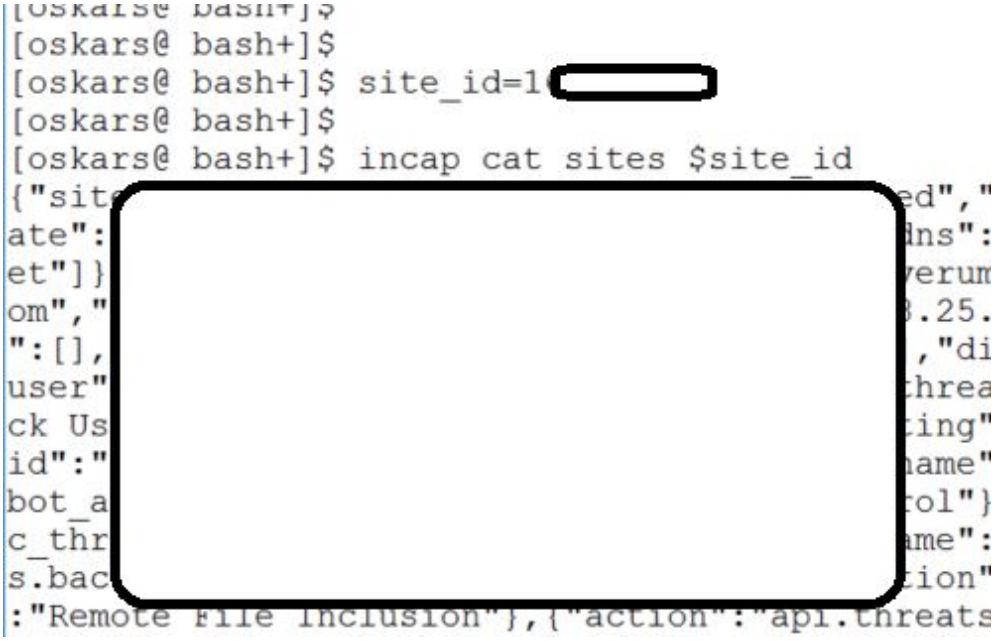
Printing original configuration json with *incap cat*

You can print the original json of Incapsula configuration using *incap cat* command
incap cat accounts will print the json of all your accounts



incap cat sites will print jsons of all files. Not showing here

incap cat sites <*site id*> or <*site name*> will print the json of that site



incap cat sites <*site id/domain*> print configuration stored locally. If you want to be 100% sure that you are not missing any changes, for example done by other people thru Incapsula web site, use *incap site status* <*site id*>. This command will download the latest configuration of the site from Incapsula API

Notes on *incap updatedb*

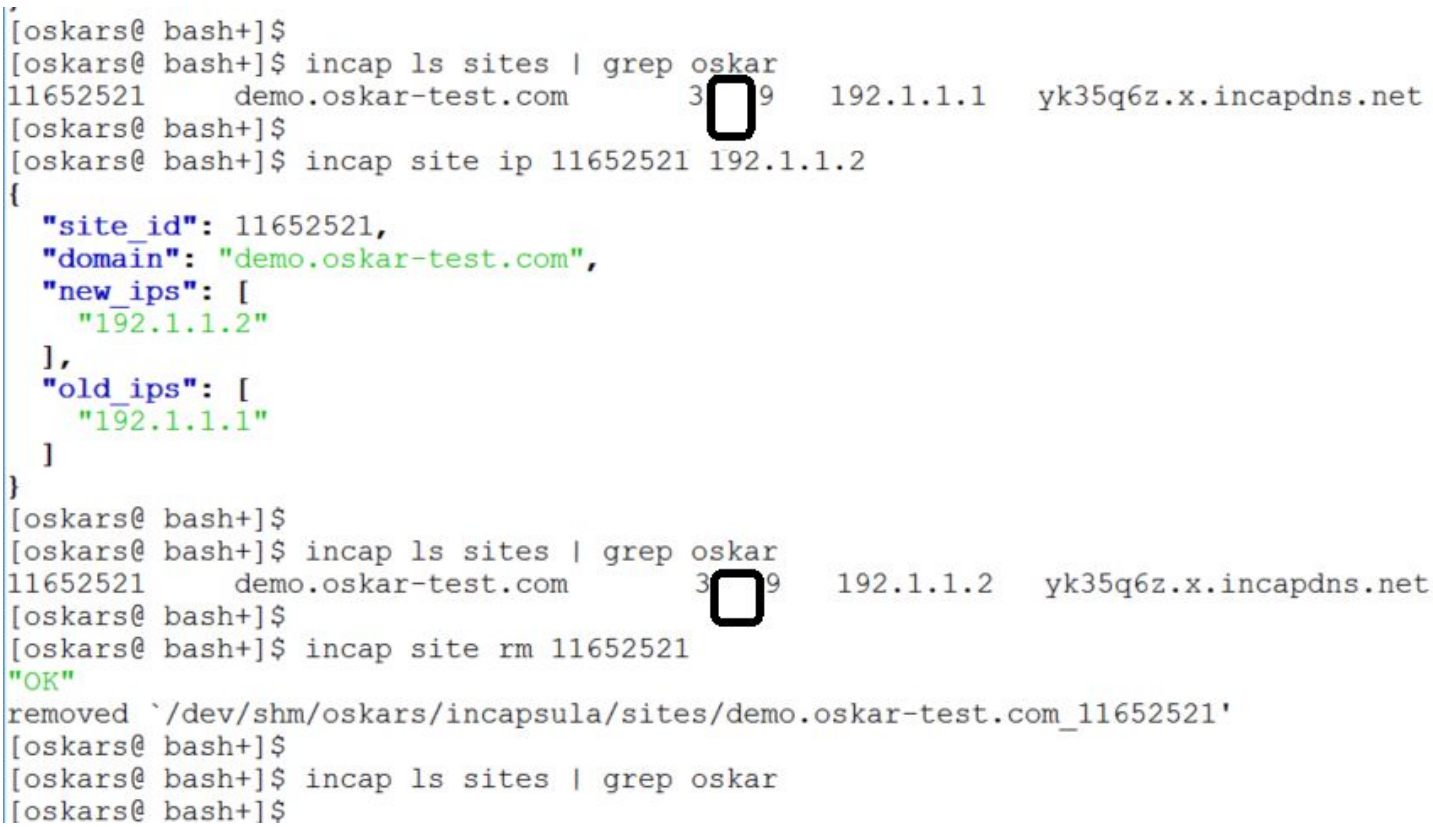
Incapsula identifies all sites by their ids and not by their names. At the moment, Incapsula API doesn't have a command that can query the site id by the site's name. Instead, you should download the entire configuration of all your sites to be able to map names to ids.

Even if such a command existed, there are still advantages to have your Incapsula configuration locally cached because this allows you to search your sites by their account or origin IP, like in the examples above, or any other parameter.

It's a good practice to sync your Incapsula configuration every once in a while. However, bash+ generally keeps your local cache updated with the following tricks.

First of all, when you create or modify a site, Incapsula API returns the updated configuration in a json format. *incap api* automatically detects if the response contains the site configuration and updates your cache.

Two, when you delete a site using command *incap site rm <site id>*, the site is deleted from your local cache as well. Below is a few examples:



Making API requests with *incap api*

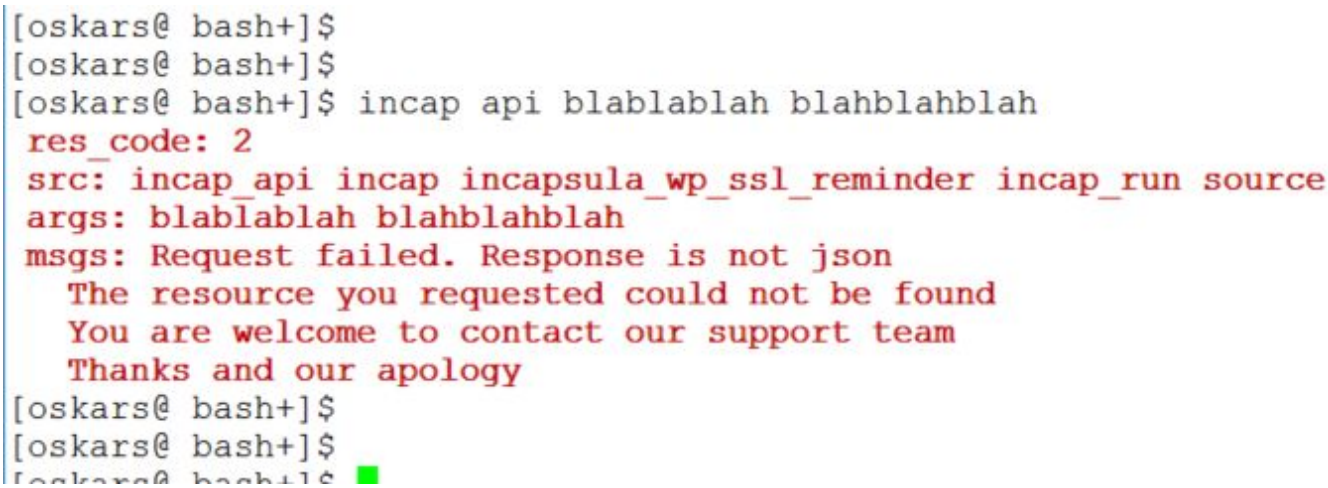
You have already seen that incap api is evoked as *incap api <post data> <url/uri>*
It's important to keep in mind that *incap api* doesn't print output on success. It only prints error messages when requests fail.

If you do need access to the output of the request, it's stored in a special reserved variable called *res*.
In the example below we access this variable to read the TXT record for ssl validation.

```
[oskars@ bash+]$
[oskars@ bash+]$ incap ls sites demo.oskar-test.com
Site ID                  = 13663571
Site Name                = demo.oskar-test.com
Site Status              = pending-dns-changes
Account ID               = 3[redacted]
Origin                   = 192.1.1.1
Cache Mode               = advanced
Block URLs               =
SSL CA                   =
SSL San                  =
SSL Detection            = true
SSL Detection Status     = ok
SSL Validation Status    =
DNS CNAME Record         = gcnz4at.x.incapdns.net
DNS A Records            =
[oskars@ bash+]$
[oskars@ bash+]$ incap site ssl 13663571
[oskars@ bash+]$
[oskars@ bash+]$ echo $res_code
0
[oskars@ bash+]$
[oskars@ bash+]$ echo $res | jq .
{
  "res": 0,
  "res_message": "OK",
  "debug_info": {
    "id-info": "9088",
    "domain_dns": "globalsign-domain-verification=cxIzH
  }
}
[oskars@ bash+]$
```

Another variable *res_code* saves the exit code of the last *incap api* call.
res_code can have the following values:
0 - Success
1 - API returned error
2 - Request failed and the error message received is not json. Most likely you misspelled the url and the response is an html page
10+ - Error codes of curl program which bash+ uses for requests. They are calculated by adding 10 to the original exit code of curl

On errors *incap api* prints res_code, the history list of the last 10 functions including the last request and the arguments of the last request



Additional commands

```
[oskars@ bash+]$
[oskars@ bash+]$ incap site id demo.oskar-test.com
13663571
[oskars@ bash+]$
[oskars@ bash+]$ incap site id 13663571
13663571
[oskars@ bash+]$
[oskars@ bash+]$ incap site purge cache 13663571
Purging site 13663571 cache...
"OK"
[oskars@ bash+]$
[oskars@ bash+]$ incap site purge hostname demo.oskar-test.com
Purging demo.oskar-test.com from hostname cache...
"OK"
[oskars@ bash+]$
```

bash+ also provides some auxiliary functions, such as *json2bash* for reformatting json in case you may find it more comfortable to work with json

```
[oskars@ bash+]$
[oskars@ bash+]$ incap site ssl 13663571
[oskars@ bash+]$
[oskars@ bash+]$ echo $res | jq .
{
  "res": 0,
  "res_message": "OK",
  "debug_info": {
    "id-info": "9088",
    "domain_dns": "globalsign-domain-verification=cxIzHZv5AyxGt-12DZAw0F2CkSmRka04Z-4M6_l28b"
  }
}
[oskars@ bash+]$
[oskars@ bash+]$ echo $res | jq . | json2bash
res                     = 0
res_message             = OK
debug_info id-info      = 9088
debug_info domain_dns   = globalsign-domain-verification=cxIzHZv5AyxGt-12DZAw0F2CkSmRka04Z-4M6_l28b
[oskars@ bash+]$
```