Freva – BUG – Basic User Guide (ALPHA VERSION)

November 19, 2016

1 Start the Evaluation System via Freva

... in the shell

To log in to the miklip system you may use ssh from any linux/unix system:

```
ssh –X user–account@research.group.shell.domain
```

The -X will allow you to connect to the remote X server (e.g. to display images). user-account should be your account you usally login with, if you still don't have one please ask the admins.

Start setting up the environment by loading the proper module (You may copy the following line as is into your shell):

```
module load freva
```

This activates the system for your current session. You might notice some other modules have been loaded.

... in the web

To log in to the research group system you may use a browser:

```
research.group.web.domain
```

The domain could be different to the shell domain, depending on how the admins set up the system. Use your normal user account.

2 Work with the Evaluation System via Freva

Freva is the all in one framework with the main features: freva –help

```
Freva
Available commands:
  --plugin        : Applies some analysis to the given data.
```

```
  --history       : provides access to the configuration history
      (use --help for more help)
  --databrowser   : Find data in the system
  --crawl_my_data: Use this command to update your projectdata.
  --esgf          : Browse ESGF data and create wget script

This is the main tool for the evaluation system.
Usage: freva --COMMAND [OPTIONS]
To get help for the individual commands use
  freva --COMMAND --help
```

The core applications are the plugin, history and databrowser — same in web and shell.

Example Table

Freva Option	Description	Example
–plugin	apply some analysis tool	freva –plugin pca input=myfile.nc outputdir=/tmp
–history	browse your history	freva –history or freva –history –plugin movieplotte
–databrowser	Search the research groups data	freva –databrowser project=cmip5 variable=tas tin
–crawl_my_data	how2put your data into database	freva –crawl_my_data –path=/PATH/2/USERDAT
–esgf	Contact the esgf	freva –esgf –download-script /tmp/download.wget

All commands have a -h or -help flag to display the commands help. Basic commands for Freva:

To ...	Shell	Web
To get the main help	freva –help	Click on Help
To list the plugins	freva –plugin	Click on Plugins
To list the plugin help	freva –plugin sometool –help	Click on a specific plugin
To see the history	freva –history	Click on History
To see the history help	freva –history –help	Read the buttons

2.1 Plugins –plugins

This is the main access to all installed analysis tools and the history. The tools are implemented by providing plug-ins to the system. For more information on how to create a plugin check the Basic Developer Guide (BDG).

Basic Usage

To get the help:

```
$ freva --plugin --help
freva --plugin [opt] query
opt:
[...]
```

To list all available analysis tools:

```
$ freva --plugin
PCA: Principal Component Analysis
...
```

The "Overview":research.group.web/plugins of tools in the framework.

To select a particular tool:

```
$ freva --plugin pca
Missing required configuration for: input, variable
```

You see here that the PCA tool is complaining because of an incomplete configuration.

To get the help of a particular tool:

```
$ freva --plugin pca --help
PCA (v3.1.0): Principal Component Analysis
Options:
areaweight      (default: False)
                Whether or not you want to have your data area
                   weighted. This is
                done per latitude with sqrt(cos(latitude)).

boots           (default: 100)
                Number of bootstraps.
[...]
input           (default: None) [mandatory]
                An arbitrary NetCDF file. There are only two
                   restrictions to your
                NetCDF file: a) Time has to be the very first
                   dimension in the
                variable you like to analyze. b) All dimensions
                   in your variable
                need to be defined as variables themselves with
                   equal names.
                Both, a) and b), are usually true.
[...]
```

Here you see the configuration parameter, its default value (None means there is no value setup), whether the configuration is mandatory ([mandatory] marking by the default value) and an explanation about the configuration parameter.

To pass the values to the tool you just need to use the key=value construct like this:

```
$ freva --plugin pca input=myfile.nc outputdir=/tmp eofs=3
[...]
```

You may even define variables in terms of other variables like the projection name above. While doing so from the shell please remember you need to escape the $ sign by using the backslash or setting the value in single quotes (no, double quotes don't work). For example:

```
$ freva --plugin pca input=myfile_\${eofs}.nc outputdir=/tmp
    eofs=3
#or
$ freva --plugin pca 'input=myfile_${eofs}.nc' outputdir=/tmp
    eofs=3
```

If you want to know more about this bash feature see this and if you want to want to know much more then take a look at this

Quoting is very important on any shell, so if you use them, be sure to know how it works. It may help you avoid losing data!

Configuring the plugins

All configurations are saved in the –history can be seen, saved, return into a command and restarted!

You may want to save the configuration of the tool:

```
$ freva --plugin pca
--save-config=/home/<user_account>/evaluation_system/config/pca
    /pca.conf
variable=tas input=myfile.nc outputdir=/tmp eofs=3
INFO:__main__:Configuration file saved in
/home/<user_account>/evaluation_system/config/pca/pca.conf
```

Note this starts the tool. To just save the configuration without starting the tool use the -n or –dry-run flag. Also note this stores the configuration in a special directory structure so the system can find it again.

You can save the configuration somewhere else:

```
$ freva --plugin pca
--save-config=/home/<user_account>/evaluation_system/config/pca
    /pca.conf
--dry-run --tool pca variable=tas input=myfile.nc outputdir=/
    tmp eofs=3
INFO:__main__:Configuration file saved in pca.conf
```

The configuration stored will be used to overwrite the default one. This is a possible usecase:

Change the defaults to suit your general needs:

```
    $ freva --plugin pca --save-config=XXX --dry-run outputdir
        =/my_output_dir shiftlats=false
```

Prepare some configurations you'll be using recurrently

```
$ freva --plugin pca --save-config=XXX --dry-run
 --config-file pca.tas.conf --tool pca variable=tas
$ freva --plugin pca --save-config=XXX --dry-run
--config-file pca.uas.conf --tool pca variable=uas
```

Scheduling

Instead of running your job directly in the terminal, you can involve the SLURM scheduler.

To run the tool murcss analyzing the variable tas the command is

```
$ freva --plugin murcss variable=tas ...
```

The execution takes a certain time (here: roughly 1 minute) and prints

```
Searching Files
Remapping Files
Calculating ensemble mean
Calculating crossvalidated mean
Calculating Anomalies
Analyzing year 2 to 9
Analyzing year 1 to 1
Analyzing year 2 to 5
Analyzing year 6 to 9
Finished.
Calculation took 63.4807469845 seconds
```

To schedule the same task you would use

```
$ freva --plugin murcss variable=tas ... --batchmode=true
```

instead. The output changes to

```
Scheduled job with history id 414
You can view the job's status with the command squeue
Your job's progress will be shown with the command
tail -f  /home/zmaw/u290038/evaluation_system/slurm/murcss/
   slurm-1437.out
```

The last line shows you the command to view the output, which is created by the tool. In this example you would type

```
$ tail -f  /home/zmaw/u290038/evaluation_system/slurm/murcss/
   slurm-1437.out
```

For jobs with a long run-time or large amounts of jobs you schould consider to schedule them and use the batch mode!

–help

```
$ freva --plugin --help
Applies some analysis to the given data.
See research group wiki for more information.

The "query" part is a key=value list used for configuring the
   tool.
It's tool dependent so check that tool help.

For Example:
    freva --plugin pca eofs=4 bias=False input=myfile.nc
       outputdir=/tmp/test

Usage: freva --plugin [options]

Options:
  -d, --debug           turn on debugging info and show stack
     trace on
                        exceptions.
  -h, --help            show this help message and exit
  --repos-version       show the version number from the
     repository
  --caption=CAPTION     sets a caption for the results
  --save                saves the configuration locally for this
     user.
  --save-config=FILE    saves the configuration at the given file
      path
  --show-config         shows the resulting configuration (
     implies dry-run).
  --scheduled-id=ID     Runs a scheduled job from database
  --dry-run             dry-run, perform no computation. This is
     used for
                        viewing and handling the configuration.
  --batchmode=BOOL      creates a SLURM job
```

2.2 History –history

To get the history in the web just click on 'History' and browse around. In the shell:

```
$ freva --history
24) pca [2013-01-14 10:46:44.575529]
<THIS MUST BE DEFINED!>.pca.<THIS MUST BE DEFINED!>.nc {u'
   normalize...
```

```
23) pca [2013-01-14 10:46:01.322760]
None.pca.None.nc {u'normalize': u'true', u'testorthog': u'true
    ', u'...
22) nclplot [2013-01-11 14:51:40.910996]
first_plot.eps {u'plot_name': u'first_plot', u'file_path': u'
    tas_Am...
21) nclplot [2013-01-11 14:44:15.297102]
first_plot.eps {u'plot_name': u'first_plot', u'file_path': u'
    tas_Am...
20) nclplot [2013-01-11 14:43:37.748200]
first_plot.eps {u'plot_name': u'first_plot', u'file_path': u'
    tas_Am...
[...]
```

It shows just the 10 latest entries, i.e. the 10 latest analysis that were performed. To create more complex queries check the help:

```
$ freva --history --help

Displays the last 10 entries with a one-line compact
    description.
The first number you see is the entry id, which you might
use to select single entries.

DATE FORMAT
    Dates can be given in "YYYY-MM-DD HH:mm:ss.n" or any less
        accurate subset of it.
    These are all valid: "2012-02-01 10:08:32.1233431",
        "2012-02-01 10:08:32",
    "2012-02-01 10:08", "2012-02-01 10", "2012-02-01",
        "2012-02", "2012".

    These are *NOT*: "01/01/2010", "10:34", "2012-20-01"

    Missing values are assumed to be the minimal allowed value.
        For example:
    "2012" == "2012-01-01 00:00:00.0"

    Please note that in the shell you need to escape spaces.
    All these are valid examples (at least for the bash shell):
    freva --history --since=2012-10-1\ 10:35
    freva --history --since=2012-10-1" "10:35'

Usage: freva --history [options]
```

```
Options:
  -d, --debug         turn on debugging info and show stack trace
      on exceptions.
  -h, --help          show this help message and exit
  --full_text         If present shows the complete configuration
      stored
  --return_command    Show freva commands belonging to the
     history entries
                      instead of the entries themself.
  --limit=N           n is the number of entries to be displayed
  --plugin=NAME       Display only entries from plugin "name"
  --since=DATE        Retrieve entries older than date (see DATE
     FORMAT)
  --until=DATE        Retrieve entries newer than date (see DATE
     FORMAT)
  --entry_ids=IDs     Select entries whose ids are in "ids" (e.g.
      entry_ids=1,2
                      or entry_ids=5)
```

You can view the configuration used at nay time and the satatus of the created files (i.e. if the files are still there or has been modified)

```
$ freva --history --plugin=pca --limit=1 --full_text
26) pca v3.1.0 [2013-01-14 10:51:26.244553]
Configuration:
     areaweight=false
          boots=100
      bootstrap=false
    eigvalscale=false
           eofs=-1
          input=test.nc
        latname=lat
   missingvalue=1e+38
      normalize=false
      outputdir=/home/user/evaluation_system/output/pca
        pcafile=test.nc.pca.tas.nc
     principals=true
     projection=test.nc.pro.tas.nc
        session=1
      shiftlats=false
     testorthog=false
        threads=7
       variable=tas
```

```
Output:
  /home/user/evaluation_system/output/pca/test.nc.pca.tas.nc (
     deleted)
```

The history offers a more direct way to re-run tools. The option return_command shows the plugin command belonging to the configuration. Here an example for the tool movieplotter:

```
freva --history --plugin=movieplotter --limit=1 --
   return_command
```

It returns:

```
freva --plugin movieplotter latlon='None' polar='None' work='/
   home/user/evaluation_system/cache/movieplotter
   /1387364295586' reverse='False' range_min='None' collage='
   False' range_max='None' earthball='False' level='0' ntasks
   ='24' input=''/database/data4researchgroup/projectdata/
   project/product/institute/model/experiment/time_frequency/
   realm/variable/ensemble/
   variable_CMORtable_model_experiment_ensemble_startdate-
   enddate.nc'' loops='0' colortable='ncl_default' animate='
   True' cacheclear='True' resolution='800' outputdir='/home/
   user/evaluation_system/output/movieplotter' secperpic='1.0'
```

This is not an handy expression, but very useful. A re-run of the tool in batch shell could be easily performed by

```
$(freva --history --plugin=movieplotter --limit=1 --
   return_command)
```

2.3 Data-Browser –databrowser

All files available on the MiKlip server are scanned and indexed in a special server (SOLR). This allows us to query the server which responds almost immediately. Because of the miklip configuration the first time you call the tool it might take up to a couple of seconds to start. After that normally you should see results within a second.

2.3.1 Help

```
freva --databrowser --help

The query is of the form key=value. <value> might use *, ? as
   wildcards or any regular expression encclosed in forward
   slashes. Depending on your shell and the symbols used,
```

```
   remeber to escape the sequences properly. The safest would
   be to enclosed those in single quotes.

For Example:
    %s project=baseline1 model=MPI-ESM-LR experiment=/
       decadal200[0-3]/
time_frequency=*hr variable='/ta|tas|vu/'

Usage: freva --databrowser [options]

Options:
  -d, --debug           turn on debugging info and show stack
     trace on
                        exceptions.
  -h, --help            show this help message and exit
  --multiversion        select not only the latest version but
     all of them
  --relevant-only       show only facets that filter results (i
     .e. >1 possible
                        values)
  --batch-size=N        Number of files to retrieve
  --count-facet-values  Show the number of files for each
     values in each facet
  --attributes          retrieve all possible attributes for
     the current
                        search instead of the files
  --all-facets          retrieve all facets (attributes &
     values) instead of
                        the files
  --facet=FACET         retrieve these facets (attributes &
     values) instead of
                        the files
```

2.3.2 Usage

The databrowser expects a list of attribute=value (or key=value) pairs. There are a few differences and many more options (explained next).
Most important is that you don't need to split the search according to the type of data you are searching for. You might as well search for files both on observations, reanalysis and model data all at the same time.

Also important is that all searches are made case insensitive (so don't worry about upper or lower casing)

You can also search for attributes themselves instead of file paths. For example you can search for the list of variables available that satisfies a certain constraint (e.g. sampled 6hr, from a certain model, etc).

Defining the search

freva –databrowser project=baseline1 variable=tas time_frequency=mon

Defining the possible values

There are many more options for defining a value for a given attribute:

```
Attribute syntax        Meaning
attribute=value         Search for files containing exactly
   that attribute
attribute=val*          Search for files containing a value for
    attribute that starts with the prefix val
attribute=*lue          Search for files containing a value for
    attribute that ends with the suffix lue
attribute=*alu*         Search for files containing a value for
    attribute that has alu somewhere
attribute=/.*alu.*/     Search for files containing a value for
    attribute that matches the given regular expression (yes!
   you might use any regular expression to find what you want.
   Check the table after this one)
attribute=value1 attribute=value2       Search for files
   containing either value1 OR value2 for the given attribute (
   note that's the same attribute twice!)
attribute1=value1 attribute2=value2     Search for files
   containing value1 for attribute1 AND value2 for attribute2
attribute_not_=value    Search for files NOT containing value
attribute_not_=value1 attribute_not_=value2     Search for
   files containing NEITHER value1 or value2
```

NOTE: When using * remember that your shell might give it a different meaning (normally it will try to match files with that name) to turn that off you can use backslash / in most shells

Regular Expressions must be given within forward slashes (/) and are match agains the whole value and not some part of it. Here's a summary (there might be more... check it!) Syntax Meaning

```
==Characters==
<any_non_special_character>     that character
        Any one character
[<any_character>]       Any one character between brackets
[<any_character>-<any_other_character>]         Any one
   character between those characters
```

```
(e.g. [a-e] is like [abcde]
==Repetitions==
*       0 or more times
+       1 or more times
{n}     exactly n times
{n,}    at least n times
{n,m}   from n to m times
RegExA|RegExB   Either RegExpA or RegExpB
==Some examples==
abc     exactly "abc"
[abc]   either "a", "b" or "c"
[abc]{3}        three characters from those given. E.g. "aaa",
   "bab" or "cab"
[abc]{2,4}      two to four characters from those given. E.g. "
   aaa", "ab" or "cccc"
[a-z]+[0-9]*    One ore more characters followed by cero or
   more number, e.g. "a",
"tas", "cfaddbze94"
```

Searching for metadata

You might as well want to now about possible values that an attribute can take after a certain search is done. For this you use the –facet flag (facets are the possible attributes that partition the result set). For example to see the time frequency (time resolution) in which reanalysis are available you might issue the following query:

```
$ freva --databrowser --facet time_frequency project=reanalysis
    time_frequency: 6hr,day,mon
```

You might also ask for more than one single facet by definig the –facet flag multiple times. For example let's also see a list of variables:

```
$ freva --databrowser --facet time_frequency --facet variable
   project=reanalysis
variable:cl,clt,evspsbl,hfls,hfss,hur,hus,pr,prc,prsn,prw,ps,
   psl,rlds,rldscs,rlus,rlut,rlutcs,rsds,rsdscs,rsdt,rsut,
   rsutcs,sfcwind,ta,tas,tauu,tauv,tro3,ts,ua,uas,va,vas,wap,zg
    time_frequency: 6hr,day,mon
```

Please note that those are not related, i.e. the values of the time_frequency facet do not correspond to any particular variable. It is like issuing to difference queries.

Also note that you can further define this as usual with a given query. For example check which files are at 6hr frequency:

```
$ freva --databrowser --facet variable project=reanalysis
   time_frequency=6hr
variable: psl,sfcwind,tas,zg
```

If you want to see how many files would return if you further select that variable (drill down query) you may add the –count-facet-values flag (simply –count will also do):

```
$ freva --databrowser --count-facet-values --facet variable
   project=reanalysis time_frequency=6hr
variable: psl (7991),sfcwind (33),tas (33),zg (131)
```

This means that there are 7991 files containing the variable psl, 33 for sfcwind, and so on. If you want to check all facets at once you may use the –all-facets flag (don't worry this is still very fast)

```
$ freva --databrowser --all-facets project=reanalysis
time_frequency=6hrcmor_table: 6hrplev
realm: atmos
data_type: reanalysis
institute: ecmwf,jma-criepi,nasa-gmao,ncep-ncar,noaa-cires
project:
time_frequency: 6hr
experiment: 20cr,cfsr,eraint,jra-25,merra,merra_testarea,ncep1,
   ncep2
variable: psl,sfcwind,tas,zg
model: cdas,cfs,geos-5,ifs,jcdas,nomads
data_structure:
ensemble: r10i1p1,r11i1p1,r12i1p1,r13i1p1,r14i1p1,r15i1p1,
   r16i1p1,r17i1p1,r18i1p1,r19i1p1,r1i1p1,r20i1p1,r21i1p1,
   r22i1p1,r23i1p1,r24i1p1,r25i1p1,r26i1p1,r27i1p1,r28i1p1,
   r29i1p1,r2i1p1,r30i1p1,r31i1p1,r32i1p1,r33i1p1,r34i1p1,
   r35i1p1,r36i1p1,r37i1p1,r38i1p1,r39i1p1,r3i1p1,r40i1p1,
   r41i1p1,r42i1p1,r43i1p1,r44i1p1,r45i1p1,r46i1p1,r47i1p1,
   r48i1p1,r49i1p1,r4i1p1,r50i1p1,r51i1p1,r52i1p1,r53i1p1,
   r54i1p1,r55i1p1,r56i1p1,r5i1p1,r6i1p1,r7i1p1,r8i1p1,r9i1p1
```

And again you can also have the –count flag:

```
$ freva --databrowser --all-facets --count project=reanalysis
   time_frequency=6hr
cmor_table: 6hrplev (8188)
realm: atmos (8188)
data_type: reanalysis (8188)
institute: ecmwf (132),jma-criepi (66),nasa-gmao (99),ncep-ncar
    (163),noaa-cires (7728)
project:
time_frequency: 6hr (8188)
experiment: 20cr (7728),cfsr (64),eraint (132),jra-25 (66),
   merra (66),merra_testarea (33),ncep1 (65),ncep2 (34)variable
```

```
   : psl (7991),sfcwind (33),tas (33),zg (131)model: cdas (99),
   cfs (64),geos-5 (99),ifs (132),jcdas (66),nomads (7728)
data_structure:
ensemble: r10i1p1 (138),r11i1p1 (138),r12i1p1 (138),r13i1p1
   (138),r14i1p1 (138),r15i1p1 (138),r16i1p1 (138),r17i1p1
   (138),r18i1p1 (138),r19i1p1 (138),r1i1p1 (598),r20i1p1(138),
r21i1p1 (138),r22i1p1 (138),r23i1p1 (138),r24i1p1 (138),r25i1p1
    (138),r26i1p1 (138),r27i1p1 (138),r28i1p1 (138),r29i1p1
   (138),r2i1p1 (138),r30i1p1 (138),r31i1p1 (138),r32i1p1 (138)
   ,r33i1p1 (138),r34i1p1 (138),r35i1p1 (138),r36i1p1 (138),
   r37i1p1 (138),r38i1p1 (138),r39i1p1 (138),r3i1p1 (138),
   r40i1p1 (138),r41i1p1 (138),r42i1p1 (138),r43i1p1 (138),
   r44i1p1 (138),r45i1p1 (138),r46i1p1 (138),r47i1p1 (138),
   r48i1p1 (138),r49i1p1 (138),r4i1p1 (138),r50i1p1 (138),
   r51i1p1 (138),r52i1p1 (138),r53i1p1 (138),r54i1p1 (138),
   r55i1p1 (138),r56i1p1 (138),r5i1p1 (138),r6i1p1 (138),r7i1p1
    (138),r8i1p1 (138),r9i1p1 (138)
```

You might have also seen that some facets are not relevant at all as they are not partitioning the resulting data (e.g. see cmor_table or data_type). You can leave them out by adding the –relevant-only flag

```
$ freva --databrowser --all-facets --count --relevant-only
   project=reanalysis
time_frequency=6hr
institute: ecmwf (132),jma-criepi (66),nasa-gmao (99),ncep-ncar
    (163),noaa-cires (7728)
experiment: 20cr (7728),cfsr (64),eraint (132),jra-25 (66),
   merra (66),merra_testarea (33),ncep1 (65),ncep2 (34)
variable: psl (7991),sfcwind (33),tas (33),zg (131)
model: cdas (99),cfs (64),geos-5 (99),ifs (132),jcdas (66),
   nomads (7728)
ensemble: r10i1p1 (138),r11i1p1 (138),r12i1p1 (138),r13i1p1
   (138),r14i1p1 (138),r15i1p1 (138),r16i1p1 (138),r17i1p1
   (138),r18i1p1 (138),r19i1p1 (138),r1i1p1 (598),r20i1p1 (138)
   ,r21i1p1 (138),r22i1p1 (138),r23i1p1 (138),r24i1p1 (138),
   r25i1p1 (138),r26i1p1 (138),r27i1p1 (138),r28i1p1 (138),
   r29i1p1 (138),r2i1p1 (138),r30i1p1 (138),r31i1p1 (138),
   r32i1p1 (138),r33i1p1 (138),r34i1p1 (138),r35i1p1 (138),
   r36i1p1 (138),r37i1p1 (138),r38i1p1 (138),r39i1p1 (138),
   r3i1p1 (138),r40i1p1 (138),r41i1p1 (138),r42i1p1 (138),
   r43i1p1 (138),r44i1p1 (138),r45i1p1 (138),r46i1p1 (138),
   r47i1p1 (138),r48i1p1 (138),r49i1p1 (138),r4i1p1 (138),
   r50i1p1 (138),r51i1p1 (138),r52i1p1 (138),r53i1p1 (138),
```

```
    r54i1p1 (138),r55i1p1 (138),r56i1p1 (138),r5i1p1 (138),
    r6i1p1 (138),r7i1p1 (138),r8i1p1 (138),r9i1p1 (138)
```

If you try to retrieve all variables stored (remember there are over +2.100.000 files!) you'll notice an ellipses (...) ath the end of the list:

```
$ freva --databrowser --facet variable
variable: abs550aer, ageice, agessc, albisccp, arag, areacella,
    areacello, bacc, baresoilfrac, basin, bddtalk, bddtdic, bddtdife,
    bddtdin, bddtdip, bddtdisi, bfe, bmelt, bsi, burntarea, c3pftfrac,
    c4pftfrac, calc, ccb, cct, ccwd, cdnc, cfad2lidarsr532, cfaddbze94,
    cfadlidarsr532, cfc11, cfc113global, cfc11global, cfc12global,
    ch4, ch4global, chl, chlcalc, chldiat, chldiaz, chlmisc, chlpico, ci
    , cl, clc, clcalipso, clcalipso2, clccalipso, cldnci, cldncl, cldnvi
    , cleaf, clhcalipso, cli, clic, clis, clisccp, clitter, clitterabove
    , clitterbelow, clivi, cllcalipso, clmcalipso, clrcalipso, cls, clt
    , cltc, cltcalipso, cltisccp, cltnobs, cltstddev, clw, clwc, clws,
    clwvi, cmisc, co2, co2mass, co3, co3satarag, co3satcalc, concaerh2o
    , concbb, concbc, conccn, concdms, concdust, concnh4, concno3,
    concoa, concpoa, concso2, concso4, concsoa, concss, cproduct, croot
    , cropfrac, csoil, csoilfast ...
```

This means there are more results than those being shown here. We limit the results to 100 for usability sake. If you still think this is a bug instead of a terrific feature, then you might use a special search word to change this facet.limit. That's the number of results that will be retrieved. Setting it to -1 retrieves just everything... be aware that make cause some problems if you don't know what you are doing (well sometimes it might also cause problems if you do... so use with discretion)

```
$ freva --databrowser --facet variable facet.limit=-1
variable:
alot!
```

By the way, do you want to count them? Those are 619 variables!

```
$ freva --databrowser --facet variable facet.limit=-1 | tr ','
    '\n' | wc -l
619
```

Bash auto completion

And if that's not awesome enough (I know it never is), then try the bash auto-completion. If you are using bash, everything is already setup when you issued the 'module load freva' command. Whenever you hit tab the word will be completed to the longest unique string that matches your previous input. A second tab will bring up a list of all possible completions after that.

For example (¡TAB¿ denotes presing the tab key):

```
freva --databrowser project=base<TAB>
```

results in

```
freva --databrowser project=baseline
```

Now pressing ¡TAB¿ again will show all other possibilities:

```
$ freva --databrowser project=baseline<TAB>
baseline0   baseline1
```

But flags are not the only thing being populated, it also work on atributes:

```
$ freva --databrowser <TAB><TAB>
cmor_table=       ensemble=         institute=        project=
           time_frequency=
data_type=        experiment=       model=            realm=
             variable=
```

... and of course values:

```
$ freva --databrowser institute=m<TAB><TAB>
miroc   mohc    mpi-m   mri
```

And (yes! That wasn't all) this is also query aware:

```
$ freva --databrowser institute=<TAB><TAB>
bcc             csiro-bom       and so on
```

```
$ freva --databrowser project=reanalysis institute=<TAB><TAB>
ecmwf       jma-criepi   nasa-gmao    ncep-ncar    noaa-cires
```

Note that if you mix flags this might not work as intended (or not at all).

2.4 –crawl_my_data

Per default it is loading the users "projectdata" directory:
/research/database/data4project/projectdata/user-account

Help

```
freva --crawl_my_data --help
Use this command to update your projectdata.

Usage: freva --crawl_my_data [options]

Options:
```

```
  -d, --debug  turn on debugging info and show stack trace on
     exceptions.
  -h, --help   show this help message and exit
  --path=PATH  crawl the given directory
```

Usage

```
freva --crawl_my_data
```

would crawl all data you have in /research/database/data4project/projectdata/user-account

When you have a lot of data in your directory, it could be worth it, to take just a sub-directory. This gets much faster, when less data.

EXAMPLE: You've put in a new decadal experiment and just want to add this

```
freva --crawl_my_data --path
/research/database/data4project/projectdata/user-account/output
    /MPI-M/MPI-ESM-LR/dec08o2000/
```

2.5 –esgf

The search syntax is defined here: http://www.esgf.org/wiki/ESGF_Search_REST_API

It has been simplified to be used from the command line and resemble freva –databrowser as clos as possible. But the two commands rely on different backends which have different query possibilities.

Help

```
The query is of the form key=value. the key might be repeated
   and/or negated with the
'_not_' suffix (e.g. model_not_=MPI-ESM-LR experiment=
   decadal2000 experiment=decadal2001)

Simple query:
    freva --esgf model=MPI-ESM-LR experiment=decadal2001
       variable=tas distrib=False

The search API is described here: http://www.esgf.org/wiki/
   ESGF_Search_REST_API
Some special query keys:
distrib: (*true*, false) search globally or only at DKRZ (MPI
   data and replicas)
```

```
latest : (true, false, *unset*) search for the latest version,
   older ones or all.
replica: (true, false, *unset*) search only for replicas, non-
   replicas, or all.

Usage: freva --esgf [options]

Options:
  -d, --debug           turn on debugging info and show stack
     trace on
                        exceptions.
  -h, --help            show this help message and exit
  --datasets            List the name of the datasets instead
     of showing the
                        urls.
  --show-facet=FACET    <list> List all values for the given
     facet (might be
                        defined multiple times). The results
                           show the possible
                        values of the selected facet according
                           to the given
                        constraints and the number of *datasets
                           * (not files)
                        that selecting such value as a
                           constraint will result
                        (faceted search)
  --opendap             List the name of the datasets instead
     of showing the
                        urls.
  --gridftp             Show Opendap endpoints instead of the
     http default
                        ones (or skip them if none found)
  --download-script=FILE
                       <file> Download wget_script for getting
                           the files
                        instead of displaying anything (only
                           http)
  --query=QUERY         <list> Display results from <list>
     queried fields
```

Usage

If you need some files: first check if they are there and how many they are:

```
$ freva --esgf project=CMIP5 experiment=decadal{1960..1965}
   variable=tas distrib=false latest=true | wc -l
278
```

You can check those urls by just not piping the result to wc (word count)

```
$ freva --esgf project=CMIP5 experiment=decadal{1960..1965}
   variable=tas distrib=false latest=true
http://cmip3.dkrz.de/thredds/fileServer/cmip5/output1/CCCma/
   CanCM4/decadal1965/day/atmos/day/r10i1p1/v20120531/tas/
   tas_day_CanCM4_decadal1965_r10i1p1_19660101-19751231.nc
http://cmip3.dkrz.de/thredds/fileServer/cmip5/output1/CCCma/
   CanCM4/decadal1965/day/atmos/day/r10i2p1/v20120531/tas/
   tas_day_CanCM4_decadal1965_r10i2p1_19660101-19751231.nc
...
```

And you can get the wget script, a bash script written around wget to simplify data download using this:

```
$ freva --esgf --download-script /tmp/scrip.wget  project=CMIP5
    experiment=decadal{1960..1965} variable=tas distrib=false
   latest=true
Download script successfully saved to /tmp/scrip.wget
```

By the way, the search looked for all files stored locally at DKRZ (distrib=false) holding the latest version (latest=true) of the variable tas (variable=tas) for the experiments decadal1960 to decadal1965 (this is a bash construct and not part of the search api!)