UNICAGE QUICK GUIDE

With focus on the Unicage Open Version

Contents

1. HOW TO INSTALL UNICAGE OPEN VERSION

Install

It's really simple to install the Unicage Open Version, you just need to follow the following steps:

1) Download and extract the repository from Github (https://github.com/Unicage-Portugal/Unicage-Open-Version).
2) Open your terminal inside the downloaded folder and type: ***sudo make install*** This will install the Unicage commands in the default path /usr/local/bin.

And it's done! You can now use your Unicage commands simply through the command line in any folder you want.

Python Commands

In the Unicage Open Version, there are some commands that come as Python scripts. To use these commands, simply copy or move the desired scripts to your project folder. Then, to call those commands, you need to type ***python ./*** and then add the command name and its respective syntax in front. For example, if you want to use the selr command, you type ***python ./selr*** and then add the respective fields in front.

Uninstall

If you want to remove the Unicage Open Version from your computer, just go to the directory containing the data you downloaded from Github, open your terminal and type: ***sudo make uninstall***

2. UNICAGE AND IT'S METHODOLOGY

Unicage is a set of highly efficient commands that allow the user to build robust, yet flexible systems in a modular way through data processing pipelines. Unicage follows the Unix philosophy, a set of concepts and guidelines that focus on designing small but highly efficient programs and operating systems.

This philosophy can be summarized in 8 core topics:

- Small is beautiful.
- One program (command) should only do one thing.
- Prototyping should be as fast as possible.
- Portability takes precedence over efficiency.
- Data is stored as plain text.
- Commands are used as "levers" (can be combined & reused).
- Applications are written in shell script.
- All programs are designed as filters (pipes).

2.1. Why is the processing speed so fast?

Each command of the Unicage Enterprise version is written in C language, and the input/output buffer, memory manipulation and calculation algorithm have been designed to allow high-speed processing.

The Shell uses kernel functions directly. By removing middleware there is no processing. Unicage Shell Script programming methodology avoids slow variable type programming and functional programming, and it follows the data flow programming that takes advantage of the processing speed of each command.

In Unicage Shell programming we organize the data in advance for increased performance. Unicage developed high-speed commands for complex sorting.

References:

https://dl.acm.org/doi/10.1145/3136014.3136031
https://benchmarksgame-team.pages.debian.net/benchmarksgame/
https://github.com/niklas-heer/speed-comparison
http://www.hildstrom.com/projects/langcomp/index.html
http://attractivechaos.github.io/plb/

2.2. Comparison with mainstream technologies

Performance benchmark vs mainstream data storage and data processing frameworks (Spark, SparkSQL, Kudu/ HDFS, Hadoop)
Research studies in prestigious higher education institutions both in the United States (MIT), Japan (Kanazawa University) and Europe (IST Lisbon) show results ranging from 3 to 50 times faster.

Gains in development productivity
Unicage provides a significant reduction of lines of code (depending on the language to be converted, for example Cobol application re-writing is a 20:1 ratio). Unicage is also easy to read and understand and provides easily measurable auditing to the usage of the system.

2.3. How is the data organized at the storage level (column, row)?

Data is stored in a UNIX file system as a regular flat text file format. The Unicage methodology consists of the way data is organized and then processed by our proprietary commands. The uniqueness of the solution lies on its ability to utilize flat text files instead of requiring middleware or relational database engines that decrease the speed of execution.

2.4. How is the data distributed across data units?

Parallel processing is done on a master-slave model where flat text files are divided according to the logic of the script across all the available nodes. Those nodes execute the script and finally they are merged and consolidated into a new text file with the result of the execution.

2.5. Data file management (levels)

Unicage organizes data files into business units along five levels ('OSAHO' – Unicage's etiquette of data file management)

- Level 1 (event data)
- Level 2 (confirmed data)
- Level 3 (organized data)
- Level 4 (application reference data
- Level 5 (application output data)

Unicage does not require the existence of any special file management software

2.6. Interoperability, locking, compression and memory management

Interoperability
Unicage is based on UNIX fundamentals. Interoperability through frameworks such as Tivoli (IBM) or JP1 (Hitachi).

Locking
Unicage does not require locking unless a concurrent situation is present. For those purposes, the command "*ulock*".

Compression
Unicage is compatible with multiple compression tools. For example .gz .Z is used for data compression.

Memory Management
Unicage processes data based on streaming, which decreases the memory usage comparatively to technologies such as java or python. Managing memory is accomplished by Unix commands.

2.7. Security (authentication, authorization and encryption)

Unicage utilizes segregation mechanisms of the underlying UNIX Operating System: filesystem permissions, memory stack protection and role-based access controls.

Encryption can be achieved on several levels - either native filesystem encryption mechanisms (F2FS in Linux or ZFS in BSD/UNIX) or 3rd party mechanisms offered by several vendors. Self-encrypting disk is also a possibility as Unicage just uses the Operating System POSIX infrastructure to access the data storage.

Security can be increased through File checksumming tools (such as native capacity or products like Tripwire) and rule-based firewall.

A typical Linux/UNIX node running Unicage will only need SSH as an open port - this can be ensured by service minimization (disabling and/or uninstalling unnecessary services). This is implemented by a host-based firewall, which will permit access only from known hosts. SSH access is restricted to a number of known users (no Administration/Super User access is conceded).

Essential configuration files are then protected by checksumming to ensure no alteration of content.

File systems can be encrypted to prevent data loss and processes will run only with needed privileges inside protected memory.

2.8. UNIX parallel processing

UNIX is a multi-user, multi-tasking OS. You can run multiple jobs for multiple users at the same time.

- Parallel processing commands used:
- Specify “& (background)“ when the job starts, to parallelize the job
- “bg” or ”fg” commands switch between parallelizing and sequencing along the processing
- “nice“ command changes the priority of parallel processing
- ”stop“ or ”kill” commands interrupt or stop the job
- “jobs” or “ps” or “tree” commands allow monitoring the parallel processing

Above mentioned job control commands, allow for writing a shell script to perform parallel processing in any number of processes.

2.9. Partitioning, indexing and concurrency

Partitioning
There are no special requirements on partitioning. Nodes can be independent servers or virtualized (i.e Docker). The only recommendation we provide is to leave 10% disk space available for our data operations inside the disk.

Indexing
There are no special requirements for indexing as everything is executed on the UNIX file system as a text file.

Concurrency
Unicage is based on UNIX fundamentals where multithreading, concurrent and exclusive processes are allowed. Our suite of commands contains blocking commands as well as atomic writing.

2.10. Scalability and workload management support

Scalability
Unicage scales quasi-linearly with extra hardware. The cluster version commands provide an automatic map and reduce process.

Workload Management support
There are multiple frameworks that control UNIX processes. As an example, for general workload management commands such as “ulimit” or “nice” can be used. Enforcing detail management usually is handled by sub-OS functions such as “cgroup” or “jail”.

3. COMMANDS AND EXAMPLES

In this section, we give you examples and descriptions of the most common and most used Unicage commands that come with the Unicage Open Version.

self

Allows you to reorder fields, select substrings from fields and/or filter content in a file.
By using ***-d*** option, you can execute the command on a string.

self field1 field2 … fieldN <file>

self -d field1 field2 … fieldN <string>

sample

```
002 023 A 20210103
001 025 C 20210103
003 012 B 20210105
002 022 C 20210102
```

self 3 1 2 sample	self 3 4 sample	self 3 1 4.1.4 4.5.2 4.7
A 002 023	A 20210103	A 002 2021 01 03
C 001 025	C 20210103	C 001 2021 01 03
B 003 012	B 20210105	B 003 2021 01 05
C 002 022	C 20210102	C 002 2021 01 02

delf

Deletes fields from a file.

delf f1 f2 … fN <file>

sample

002 023 A 20210103

001 025 C 20210103

003 012 B 20210105

002 022 C 20210102

delf 2 sample	delf 2/3 sample
002 A 20210103	002 20210103
001 C 20210103	001 20210103
003 B 20210105	003 20210105
002 C 20210102	002 20210102

selr

Selects the rows of the file that match a given string in a specific field.

selr <field> <string> <file>

NOTE: To use this command in the Unicage Open Version, you must use the respective Python script.

sample

002 023 A 20210103

001 025 C 20210103

003 012 B 20210105

002 022 C 20210102

python ./selr 3 C sample	python ./selr 1 002 sample
001 025 C 20210103	002 023 A 20210103
002 022 C 20210102	002 022 C 20210102

delr

Deletes the rows of the file that match a given string in a specific field.

delr <field> <string> <file>

NOTE: To use this command in the Unicage Open Version, you must use the respective Python script.

sample

```
002 023 A 20210103
001 025 C 20210103
003 012 B 20210105
002 022 C 20210102
```

```
python ./delr 3 C sample
002 023 A 20210103
003 012 B 20210105
```

```
python ./delr 1 002 sample
001 025 C 20210103
003 012 B 20210105
```

sm2

Sums the elements of specified fields based on the key-fields chosen by the user. The first two arguments correspond to the range of the fields the key occupies, whilst the last two arguments correspond to the range of fields that will be summed.
By using the ***+count*** option, you can obtain the total number of records that were summed.

sm2 <key1> <key2> <field1> <field2> <file>

sample_sales

```
001 Agata 20110405 32 1600
001 Agata 20110406 24 1200
001 Agata 20110407 49 2450
002 Tony 20110405 102 5100
002 Tony 20110406 98 4900
002 Tony 20110407 121 6050
```

sm2 1 2 4 5 sample_sales	sm2 ***+count*** 1 2 4 5 sample_sales
001 Agata 105 5250 002 Tony 321 16050	001 Agata 3 105 5250 002 Tony 3 321 16050

sm4

Sums the elements of specified fields based on the key-fields chosen by the user, displaying the subtotals. The first two arguments correspond to the range of the fields the key occupies, the third and fourth arguments correspond to the range of fields to be ignored, and the last two arguments correspond to the range of fields that will be summed.

sm4 <key1> <key2> <field_to_ignore1> <field_to_ignore_2> <field_to_sum1> <field_to_sum2> <file>

sample_sales

```
001 Agata 20110405 32 1600
001 Agata 20110406 24 1200
001 Agata 20110407 49 2450
002 Tony 20110405 102 5100
002 Tony 20110406 98 4900
002 Tony 20110407 121 6050
```

```
sm4 1 2 3 3 4 5 sample_sales

001 Agata 20110405 32 1600
001 Agata 20110406 24 1200
001 Agata 20110407 49 2450
001 Agata @@@@@@@@ 105 5250
002 Tony 20110405 102 5100
002 Tony 20110406 98 4900
002 Tony 20110407 121 6050
002 Tony @@@@@@@@ 321 16050
```

sm5

Sums the elements of specified fields based on ignoring the key, thus obtaining a grand total. The first two arguments correspond to the range of the fields to be ignored, whilst the last two arguments correspond to the range of fields that will be summed

sm5 <field_to_ignore1> <field_to_ignore_2> <field_to_sum1> <field_to_sum2> <file>

sample_sales

```
001 Agata 20110405 32 1600
001 Agata 20110406 24 1200
001 Agata 20110407 49 2450
002 Tony 20110405 102 5100
002 Tony 20110406 98 4900
002 Tony 20110407 121 6050
```

```
sm5 1 3 4 5 sample_sales

001 Agata 20110405 32 1600
001 Agata 20110406 24 1200
001 Agata 20110407 49 2450
002 Tony 20110405 102 5100
002 Tony 20110406 98 4900
002 Tony 20110407 121 6050
@@@ @@@@@ @@@@@@@@@ 426 21300
```

ratio

Gives the percentage of a value in the field based on the overall sum of all values of that field.

ratio [key=<K>] val=<V> <file>

sample_shop_sales

```
ShopA 20210104 10 1000
ShopA 20210105 8 800
ShopA 20210106 7 700
ShopB 20210104 5 500
ShopB 20210105 7 700
ShopB 20210106 8 800
```

```
ratio key=1 val=3/4

ShopA 20210104 10 40.0 1000 40.0
ShopA 20210105 8 32.0 800 32.0
ShopA 20210106 7 28.0 700 28.0
ShopB 20210104 5 25.0 500 25.0
ShopB 20210105 7 35.0 700 35.0
ShopB 20210106 8 40.0 800 40.0
```

It is possible to set the number of decimal places by using the option ***-N***, where **N** corresponds to the number of decimal places.

```
ratio -0 key=1 val=3/4

ShopA 20210104 10 40 1000 40
ShopA 20210105 8 32 800 32
ShopA 20210106 7 28 700 28
ShopB 20210104 5 25 500 25
ShopB 20210105 7 35 700 35
ShopB 20210106 8 40 800 40
```

join0

Extracts from the Transaction File only the records that contain items present in the Master File. The items in the Transaction File need to be sorted before applying the join0 command.

join0 [+ng] key=<K> <MASTER> <TRANSACTION>

LIST_MASTER

A 001 Billy

A 002 Dilan

B 002 Dora

B 003 Anne

RESULTS_TRANS

A 001 330

A 002 450

A 003 345

B 001 320

B 002 403

B 003 409

B 004 381

```
join0 key=1/2 LIST_MASTER RESULTS_TRANS

A 001 330
A 002 450
B 002 403
B 003 409
```

By attaching the option ***+ng*** to the join0, one can extract the records that are not present in the Master File to standard output. To separate the records, redirection must be used.

```
join0 +ng key=1/2 LIST_MASTER RESULTS_TRANS > /dev/null

A 003 345
B 001 320
B 004 381
```

You can separate the records to two distinct files by doing the following:

join0 ***+ng*** key=1/2 LIST_MASTER RESULTS_TRANS ***> RESULTS_OK 2> RESULTS_NOK***

In the RESULTS_OK file, the records from the Transaction File that are present in the Master File will be stored and in RESULTS_NOK file, the records from the Transaction File that are not present in the Master file will be stored.

join1

Similar to join0, but it adds to the Transaction File additional information that is present in the Master's File fields.

join1 [+ng] key=<K> <MASTER> <TRANSACTION>

LIST_MASTER

A 001 Billy

A 002 Dilan

B 002 Dora

B 003 Anne

RESULTS_TRANS

A 001 330

A 002 450

A 003 345

B 001 320

B 002 403

B 003 409

B 004 381

join1 key=1/2 LIST_MASTER RESULTS_TRANS

A 001 Billy 330
A 002 Dilan 450
B 002 Dora 403
B 003 Anne 409

join2

Similar to join1, but it also outputs the fields that are not present in the Master File in a special format.
join2 [+ng] key=<K> <MASTER> <TRANSACTION>

LIST_MASTER

A 001 Billy

A 002 Dilan

B 002 Dora

B 003 Anne

RESULTS_TRANS

A 001 330

A 002 450

A 003 345

B 001 320

B 002 403

B 003 409

B 004 381

```
join2 key=1/2 LIST_MASTER RESULTS_TRANS

A 001 Billy 330
A 002 Dilan 450
A 003 _ 345
B 001 _ 320
B 002 Dora 403
B 003 Anne 409
B 004 _ 381
```

By default, the fields that are not present in the Master File will have a '_' in their place. However, you can change this value by using the ***+*** option and specifying the string that you want in front.

join2 [+string] [+ng] key=<K> <MASTER> <TRANSACTION>

```
join2 +other key=1/2 LIST_MASTER RESULTS_TRANS

A 001 Billy 330
A 002 Dilan 450
A 003 other 345
B 001 other 320
B 002 Dora 403
B 003 Anne 409
B 004 other 381
```

cjoin0, cjoin1, cjoin2

The cjoin0, cjoin1 and cjoin2 commands are equivalent to the join0, join1 and join2 commands, respectively. However, the cjoin commands do not require the Transaction File to be sorted.

Since the cjoin commands load the entire Master File into memory, the usage of the cjoin commands is recommended over the join commands when you have a small Master File and extremely large Transaction Files.

loopj

Links multiple files with the same key. The files need to be sorted by key.
If there are records with non-existent values in some files, by default a "0" will be placed. This character can be changed by using the option ***-d*** and specifying the character that you want in front of it.

loopj [-d<string>] num=<K> <file_1> <file_2> … <file_N>

SAMPLE1

001 Oslo

002 Lisbon

003 Paris

004 Madrid

SAMPLE2

001 50320 324.1

002 2199 4.3

003 3310 134.8

004 2201 6.4

SAMPLE3

001 NOR

002 POR

003 FRN

loopj ***-d@*** num=1 SAMPLE1 SAMPLE2 SAMPLE3

001 Oslo 50320 324.1 NOR
002 Lisbon 2199 4.3 POR
003 Paris 3310 134.8 FRN
004 Madrid 2201 6.4 @

up3

Merges two files that have the same key. Both files need to be sorted before using up3.

up3 key=<K> <file_1> <file_2>

SAMPLE1

001 Oslo

002 Lisbon

003 Paris

004 Madrid

SAMPLE2

001 50320 324.1

002 2199 4.3

003 3310 134.8

004 2201 6.4

up3 key=1 SAMPLE1 SAMPLE2

001 Oslo 50320 324.1
002 Lisbon 2199 4.3
003 Paris 3310 134.8
004 Madrid 2201 6.4

getfirst, getlast

getfirst fetches the first records with a given key in a file, while getlast fetches the last records with a given key in a file. The file needs to be sorted by the key before using these commands.

It is possible to use the ***+ng*** option to fetch all records except the first or last, depending on the command used.

getfirst [+ng] <K1> <K2> <file>

getlast [+ng] <K1> <K2> <file>

SAMPLE

```
Japan Chiba 100
Japan Chiba 90
Japan Tokyo 200
Portugal Lisbon 120
Portugal Lisbon 100
Portugal Porto 80
```

```
getfirst 1 1 SAMPLE
Japan Chiba 100
Portugal Lisbon 120
```

```
getlast 1 1 SAMPLE
Japan Tokyo 200
Portugal Porto 80
```

ctail

Removes the last N records from a file.

ctail -<N> <file>

STATES

```
001 Maine
002 Vermont
003 New_York
004 Delaware
005 Georgia
```

```
ctail -2 STATES
001 Maine
002 Vermont
003 New_York
```

count

Counts the number of records with the same key within a file.

count <K1> <K2> <file>

SAMPLE

A

A

A

B

count 1 1 SAMPLE

A 3
B 1

rank

Counts the occurrences and associates them consecutive numbers starting in 1. By using ***ref*** option, it is possible to associate the numbers based on the given key.

rank [ref=<K>] <file>

SAMPLE

001 Tony

001 Gina

001 Anne

002 Carl

003 Sara

rank SAMPLE

1 001 Tony
2 001 Gina
3 001 Anne
4 002 Carl
5 003 Sara

rank ***ref=1*** SAMPLE

1 001 Tony
2 001 Gina
3 001 Anne
1 002 Carl
1 003 Sara

map

Converts records with repeated keys into a table of records where the repeated keys are intertwined in a way that the information becomes more perceptible. In a way, it converts the records from a format of < x, y, value > into a matrix of x per y. If there are fields without values, a 0 will be placed instead. This value can be changed with the ***-m*** option and by specifying the character.

map [-m<C>] [+yarr] num=<N> <file>

SALES

```
20110201 SHOP_A 13
20110201 SHOP_B 34
20110202 SHOP_A 20
20110202 SHOP_B 18
```

```
map num=1 SALES
*            SHOP_A SHOP_B
20110201    13       34
20110202    20       18
```

The ***+yarr*** option adds an additional horizontal axis.

SALES_JAPAN

```
SHOP_A Tokyo 2016 1354 2135 1255
SHOP_A Tokyo 2017 2133 1245 4265
SHOP_B Osaka 2016 984 824 793
SHOP_B Osaka 2017 908 1193 3145
```

```
map num=2 SALES_JAPAN

*          *       *    2016   2017
SHOP_A   Tokyo     A    1354    2133
SHOP_A   Tokyo     B    2135    1245
SHOP_A   Tokyo     C    1255    4265
SHOP_B   Osaka     A    984     908
SHOP_B   Osaka     B    824     1193
SHOP_B   Osaka     C    793     3145


map +yarr num=2 SALES_JAPAN

*          *        2016  2016  2016  2017  2017  2017
*          *          a     b     c     a     b     c
SHOP_A   Tokyo      1354  2135  1255  2133  1245  4265
SHOP_B   Osaka      984   824   793   908   1193  3145
```

calsed

String substitution by using a customized version of sed. It does not support regular expressions. If the string is replaced by a "@" character, then the ***-nx*** option must be used. If you want to replace a given character C by a blank space, then the ***-s*** option must be used.

calsed [-s<C>] [-nx] <ORIGINAL_STRING> <REPLACEMENT_STRING> <file>

NOTE: To use this command in the Unicage Open Version, you must use the respective Python script.

SAMPLE

```
*Tokyo*

*Lisbon*

*Milan*
```

```
python ./calsed Milan Rome SAMPLE

*Tokyo*
*Lisbon*
*Rome*
```

```
python ./calsed -nx Milan @ SAMPLE

*Tokyo*
*Lisbon*
*@*
```

```
python ./calsed -s_ Lisbon La_Paz SAMPLE

*Tokyo*
*La Paz*
*Milan*
```

fsed

Replaces a string on a given field of each record. Supports regular expressions through the ***-e*** option.

fsed [-e] 's/<ORIGINAL_STRING>/<REPLACEMENT_STRING>/<field_number>' <file>

NOTE: To use this command in the Unicage Open Version, you must use the respective Python script.

SCORES

```
000149 Lyn 18 F 95 80 33 50

000189 Joel 19 M 70 98 55 72

000152 Bill 17 M 84 79 85 62
```

```
python ./fsed 's/0/-/1' SCORES

---149 Lyn 18 F 95 80 33 50
---189 Joel 19 M 70 98 55 72
---152 Bill 17 M 84 79 85 62
```

```
python ./fsed -e 's/[0-9]/*/3' SCORES

000149 Lyn ** F 95 80 33 50
000189 Joel ** M 70 98 55 72
000152 Bill ** M 84 79 85 62
```

keycut

Divides a file into sub-files according to the specified key. These sub-files will contain only the records with the chosen key.

keycut %<K_1> %<K_2> … %<K_N> <file>

US_DATA

```
01 Massachusetts 03 Springfield  82 0  23 84 10
01 Massachusetts 01 Boston       91 59 20 76 54
02 New_York      04 Manhattan    30 50 71 36 30
02 New_York      05 Brooklyn     78 13 44 28 51
03 New_Jersey    10 Newark       52 91 44 9  0
03 New_Jersey    12 Moorestown   95 60 35 93 76
04 Pennsylvania  13 Philadelphia 92 56 83 96 75
04 Pennsylvania  16 Hershey      45 21 24 39 03
```

```
keycut STATE_DATA%1 US_DATA
```

STATE_DATA.01

```
01 Massachusetts 03 Springfield  82 0  23 84 10
01 Massachusetts 01 Boston       91 59 20 76 54
```

STATE_DATA.03

```
03 New_Jersey    10 Newark       52 91 44 9  0
03 New_Jersey    12 Moorestown   95 60 35 93 76
```

STATE_DATA.02

```
02 New_York      04 Manhattan    30 50 71 36 30
02 New_York      05 Brooklyn     78 13 44 28 51
```

STATE_DATA.04

```
04 Pennsylvania  13 Philadelphia 92 56 83 96 75
04 Pennsylvania  16 Hershey      45 21 24 39 03
```

dayslash

Transformation of Time/Date Format.
With the ***--output*** option, the existing date will be converted to a chosen format. With the ***--input*** option, the existing date will be converted from its current format to the default format (yyyymmdd).

dayslash [option] <date_format> <field_1> <field_2> … <field_N> <file>

DATES

20210101 2021/02/01 20210301

20210102 2021/02/02 20210302

20210103 2021/02/03 20210303

dayslash ***--output*** yyyy/mm/dd 1 3 DATES

2021/01/01 2021/02/01 2021/03/01
2021/01/02 2021/02/02 2021/03/02
2021/01/03 2021/02/03 2021/03/03

dayslash ***--input*** yyyy/mm/dd 2 DATES

20210101 20210201 20210301
20210102 20210202 20210302
20210103 20210203 20210303

comma

Adds comma as numeric separator.

comma <field_1> <field_2> … <field_N> <file>

SAMPLE_NUMS

1234567890

98765

13243546

comma 1 SAMPLE_NUMS

1,234,567,890
98,765
13,243,546