CLIChart Documentation

Release 0.6.0a1

John Dickson

June 22, 2014

Contents

CLIChart provides tools for quick visualisation of tabular data on the command line by generating and displaying charts, for extracting that tabular data from text data (like log files), and for manipulating the tabular data. The emphasis here is on **quick** - the idea is provide fast ways for you to visualise the relationships in your data.

Contents:

CHAPTER 1

Introduction

If a picture is worth 1,000 words, then a chart (or graph) is worth 10,000 lines of data.

CLIChart provides tools for quick visualisation of tabular data on the command line by generating and displaying charts, for extracting that tabular data from text data (like log files), and for manipulating the tabular data. The emphasis here is on **quick** - the idea is provide fast ways for you to visualise the relationships in your data.

CLIChart can:

- Display charts in a window, save them to disk (JPEG or PNG), or both
- Accept data in comma- or whitespace-separated formats
- Read data from a file, or have it piped into its standard input
- Display XY line charts, with the X axis based on dates, times or values. The Y axis must be simple values (integer or decimal)
- Be used interactively, or driven via scripts
- Generate summary data based on counts, averages, minimum, maximum of input fields
- Generate summary data based on discrete values
- Generate aggregates and histograms from tabular data
- Merge tabular data from multiple sources
- Operate in CLI server mode, generating 1 or more charts based on commands passed via standard in (either from another script, or from a command file).

Probably the easiest way to get a feel for what CLIChart can do, and whether it's of interest to you, is to look at the *Quick Start Guide*.

CLIChart was originally developed for interactive charting of data from server log files, so its primary audience is probably software developers and system administrators.

The project consists of the following tools:

- clichart - generates charts from tabular data
- cliserverlib - a Python driver for the CLI server mode of clichart
- linestats and discretestats - generate tabular data from logs or other text data
- mid - extracts parts of files or other data
- merge - merges tabular data from 2 or more sources
- aggregate - aggregates (or summarises) data from other tabular data

- histogram - generates histograms (frequency spread) from other tabular data.

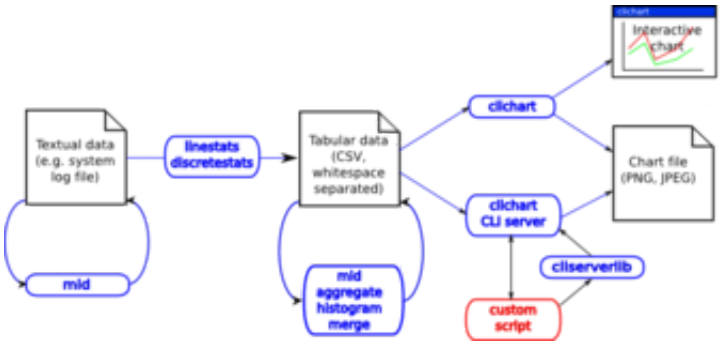
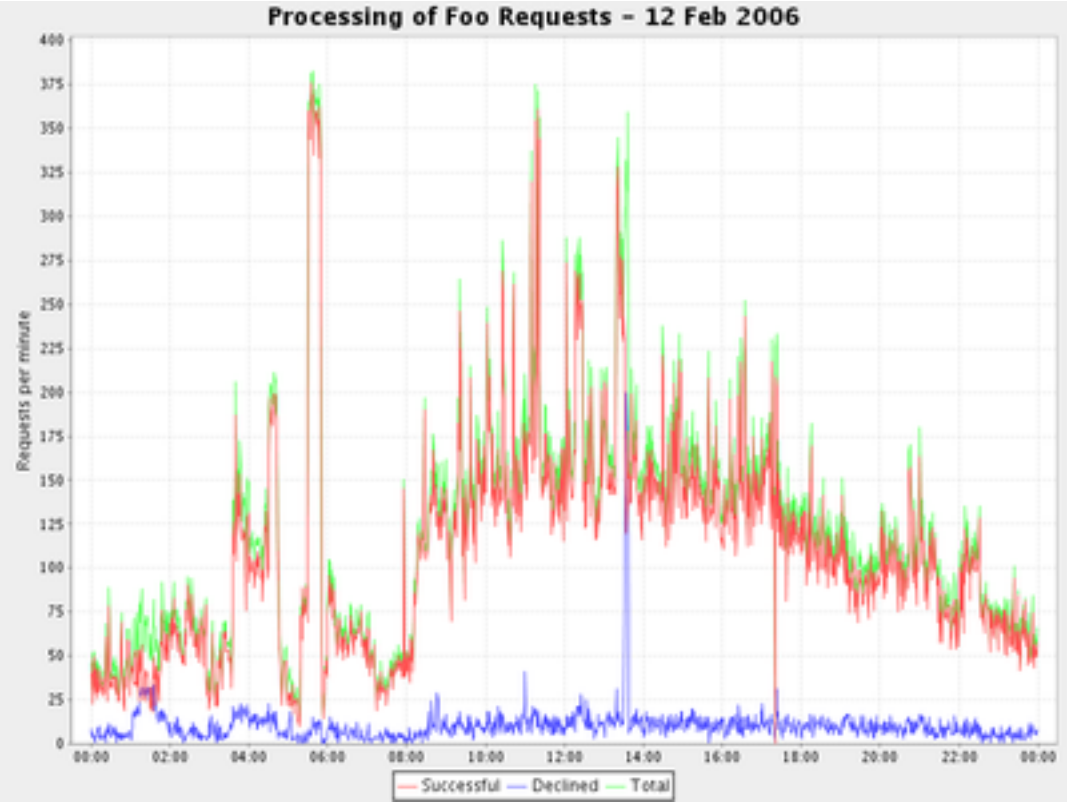


Figure 1.1: Tools in the CLIChart package - click for the full-sized diagram

NOTE: CLIChart is no longer under active development.

1.1 Sample Output

Here's a sample chart generated by CLIChart, from a CSV file extracted from data in a system log (click the image to see the full-size chart).

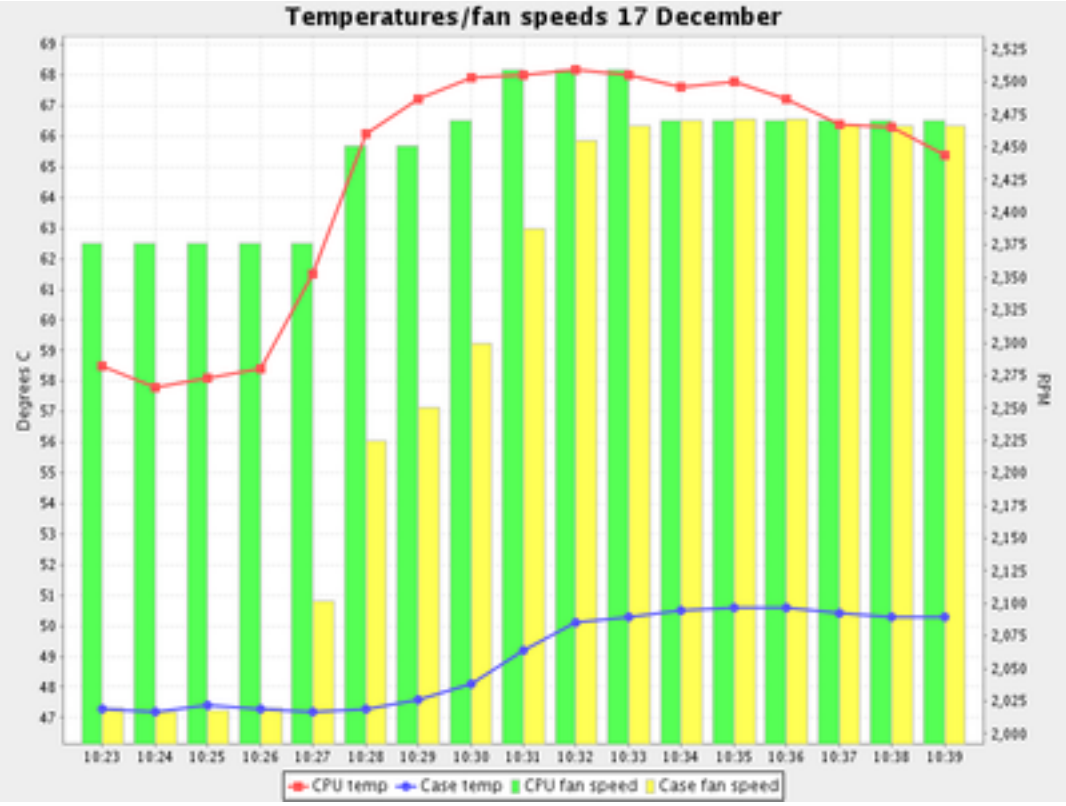


The command-line required to generate the chart was:

```
clichart -fcl 0,1,2,3 -t "Processing of Foo Requests - 12 Feb 2006" \
        -y "Requests per minute"  system.log.2006-02-12.fooRates.csv
```

And bear in mind that most of the command-line is titles and other prettiness, so it's simple and quick.

Here's a more involved chart, including a second Y axis displayed as a bar.



1.2 Modes of Operation - Using CLIChart

There are three main ways in which you might use clichart (the charting tool in CLIChart):

Automated In this mode, you have already prepared the tabular data, and you want to use *clichart* to create and save charts without human intervention. You can call clichart from shell scripts or batch files, or directly from Java if you like. The CLI server mode of *clichart* is made for such use, since it makes the process much more efficient. We use this mode to automatically generate charts for summary web pages from the daily logs of a number of systems we operate - things like transaction rates over time, memory and thread use every minute etc.

Interactive with pre-existing data As in the automated mode above, the tabular data already exists. Your aim is to use *clichart* to look for patterns or relationships in it. You feed the data to clichart on the command line, specify how it should build the chart, and then view the results in a window.

Interactive without pre-existing data In this case, you're examining some data source that isn't tabular, e.g. a server log file. You want to use *clichart* to examine it, e.g. to look for the causes of last Thursday's spectacular system meltdown. Before you can chart the data, you first need to extract the appropriate data from the source into a tabular form, and maybe massage it. This tends to be a highly iterative process - create a command line to extract and chart some data from the log, then based on the look of the chart modify the data extraction command line and repeat. The *Quick Start Guide* has some good examples.

When you have pre-existing data you can use a spreadsheet to do your charting - clichart will be faster, but the spreadsheet will be more flexible. But clichart really comes into its own for the first and third modes.

When you're mining a log file (the third mode), you'll need to use tools to extract and massage the data before charting it - this is what *linestats*, *discretestats* and *mid* are for. As well, you may not know in advance exactly what data you're looking for (typical when investigating a system problem), so this tends to be an iterative process: use a command line to generate a chart, examine the chart, tweak the command line to extract better or different data, and repeat.

1.3 Tool Documentation

1.3.1 clichart

clichart is the main program, used for generating, displaying and saving charts from tabular data

1.3.2 cliserverlib

cliserverlib is a Python driver for the CLI server mode of clichart. Use this when you have a Python script which needs to generate more than one chart (or when you want to do the same in some other language - this will provide an example of how to do it)

1.3.3 linestats

linestats is a powerful utility to generate statistical data from textual input such as a log file. It's particularly useful for extracting rates of occurrence of particular messages, e.g. transactions per second, URL accesses per minute, or for accumulating statistics (minimum, average, maximum etc.) of different values per minute.

1.3.4 discretestats

discretestats is another utility to generate statistical data from textual input. It differs from linestats in that its purpose is to count the occurrence of each discrete value in a field, and group these by key value. A good example is extracting the number of info, warning and error messages per minute in a log.

1.3.5 mid

mid is the steroidally-enhanced child of the venerable Unix head and tail utilities. It's there to extract ranges of lines from a file in the most convenient way possible. It's most useful to pull out parts of the data, in order to drill into the detail.

1.3.6 merge

merge is used to merge tabular data from 2 or more files, based on key values in each file. A typical use is to combine data from 2 sources so they can be shown on the same chart.

1.3.7 aggregate

aggregate is used to aggregate or summarise existing tabular data - you end up with 1 line of summary data for each input file. A typical use is generating data to show long-term trends, where you already have a number of tabular data files covering shorter periods.

1.3.8 histogram

histogram is used to generate a histogram (or frequency representation) of a single column from existing tabular data. You specify how many frequency intervals to split the range into, and the output is one line per interval, including the number of data values falling within that interval. A typical use would be to generate a chart showing the spread and frequency of response times from a server.

1.4 Using clichart as a Library

If you're using Java (or a scripting language running in a JVM, e.g. Jython, JRuby, Groovy etc.), then you can use clichart as a library.

You can generate the Javadoc for clichart by checking out the source code, and running:

```
ant javadoc
```

The Javadoc will be under `target/java/javadoc/`.

Alternatively, if you're using Python, you can use *cliserverlib*, particularly if you want to generate more than one chart.

1.5 Contacting Me

My name's John Dickson. I'm an architect for server-side Java systems, and I seem to have spent far too much of my life looking at logs trying to understand what the system was doing at particular moments. CLIChart grew out of that experience.

You can get in touch with me at the email address: 'captsens at gmail dot com'. I'd love to hear whether CLIChart scratches your itch, and if not why not.

1.6 Thanks and Acknowledgements

CLIChart uses several great open source libraries to do its job:

- The amazing JFreeChart, to generate and display the charts (LGPL)
- The Apache Commons CLI library, for parsing the command-line arguments (Apache licence)
- The JavaCSV library for parsing CSV data (LGPL).

Thanks to all the developers involved for making their work available to others.

1.7 Licence

CLIChart is licenced under the GNU Lesser (or Library) Public Licence. See the LICENCE.txt file for details.

CHAPTER 2

Quick Start Guide

2.1 Introduction

In the immortal words of the Python README, "If you don't read instructions, congratulations on getting this far. :-)".

Many of these examples use data in the `samples` directory.

2.2 Installation

1. Make sure you have Java installed
2. Make sure you have Python installed
3. Download the CLIChart zip file from the releases page on Github, and extract to a suitable directory
4. If you want the tools to be installed in your path (recommended):
    - Install easy_install in the setuptools module
    - Use `easy_install clichart-x.y.z.egg` to install (insert your version number)
        - Note that on Linux you'll have to be root (or use sudo)

Test your installation by typing `clichart -h` at a command prompt - this should give you the help screen.

See the installation page for further information.

2.3 Charting Pre-existing Data

Before you start, make sure that the CLIChart `bin` directory is in your path.

- On Windows: `set PATH=<pathToClichart\bin;%PATH%`
- On Linux/Unix: `export PATH=<pathToClichart/bin:$PATH`

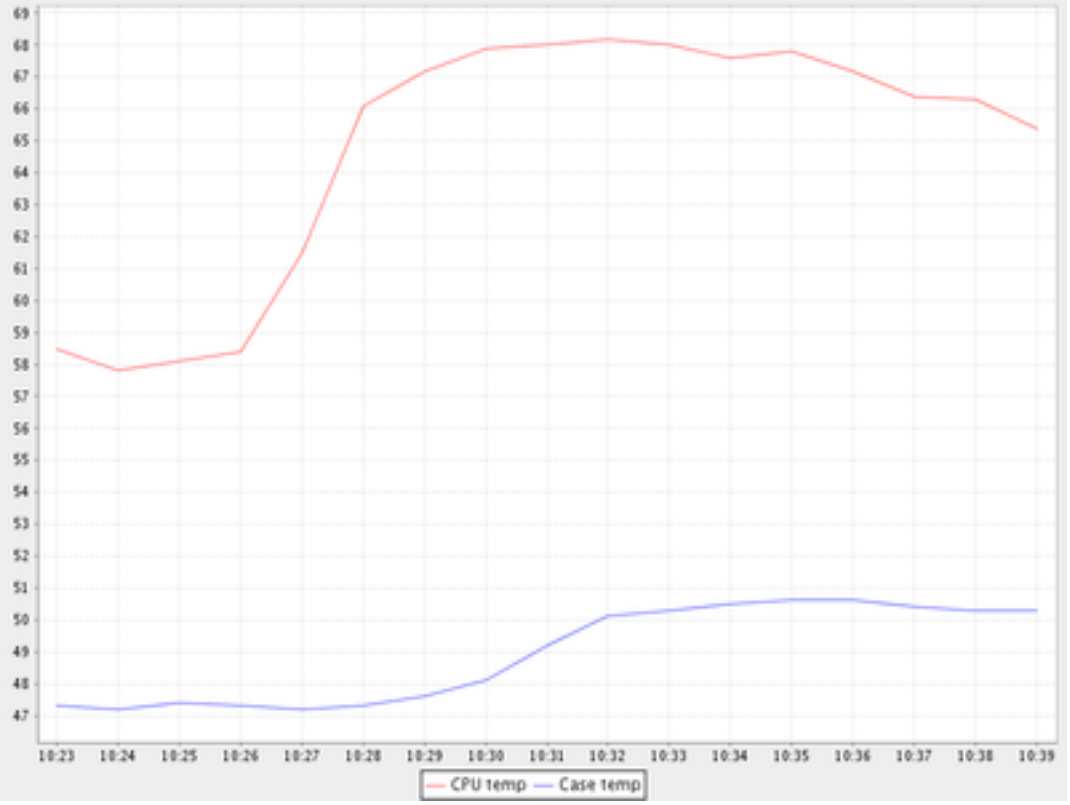
In the samples directory, you'll find `SystemTemps.csv`, which contains temperatures and fan speeds from within a computer, recorded over 15 or so minutes.

2.3.1 First Step - A Basic Chart

Let's generate a very basic chart to see what the CPU and case temperatures did over time. This is an example of the second mode of operation for clichart - interactive with pre-existing tabular data.

```
clichart -f -c -l 0,1,2 samples/SystemTemps.csv
```

And the result is a window showing the following graph (click to see the full-size chart):



Notes:

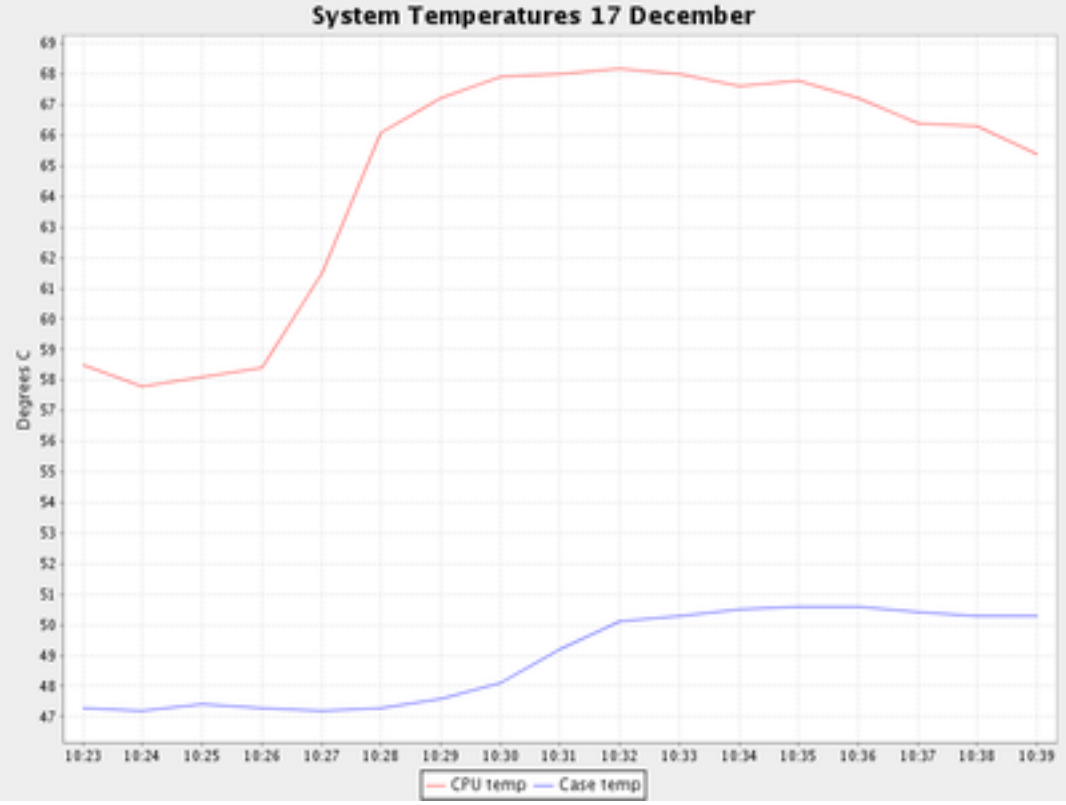
- The options used are:
  - `-f` - The first row of data is column titles or headers
  - `-c` - The data is CSV (comma-separated value)
  - `-l 0,1,2` - The date or time is in column 0, and we want to plot the data in columns 1 and 2
- Explore the top and popup menus. The popup menu in particular allows control over most aspects of the chart.

2.3.2 Adding Titles and Saving to File

Now, we'll embellish the chart a little, and instead of displaying it in a window we'll have it saved to a PNG file:

```
clichart -f -c -l 0,1,2 -t "System Temperatures 17 December" -y "Degrees C" \
        -o SystemTemps.png samples/SystemTemps.csv
```

The extra options are the chart title (`-t`), a title for the Y axis (`-y`), and the output filename for the generated chart (`-o`). The current directory now contains a file called `SystemTemps.png`, which looks like this:



2.3.3 Management-Friendly Charts - The Second Axis

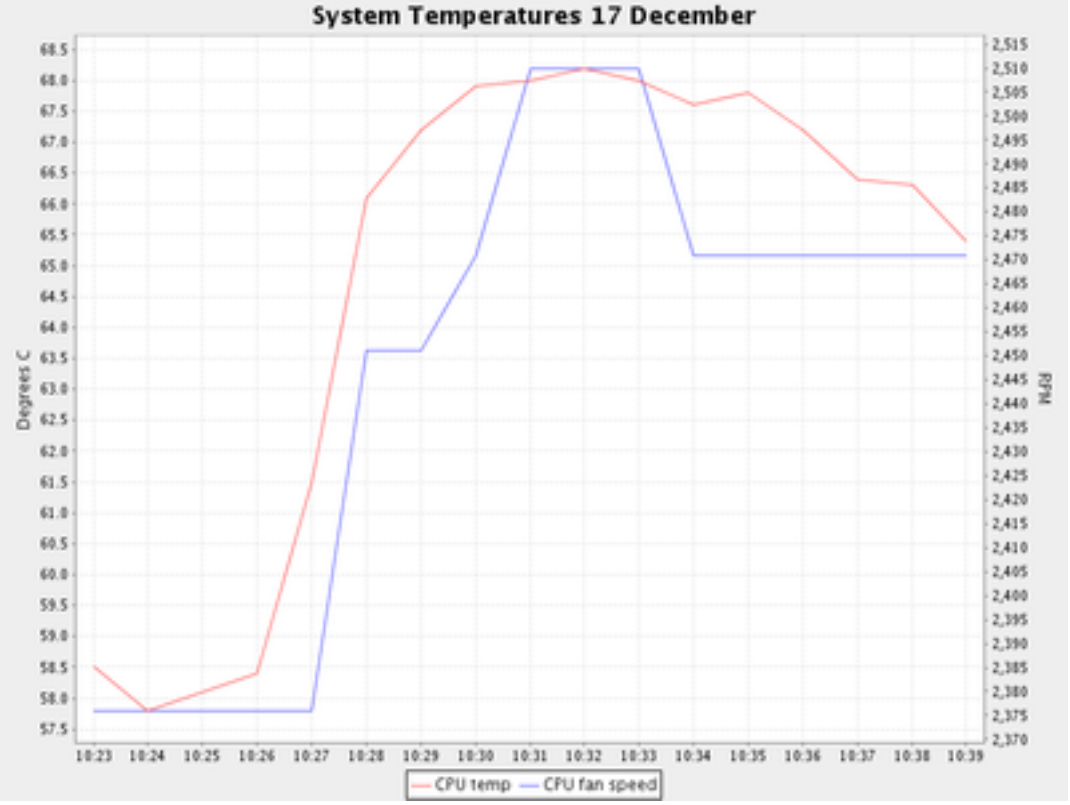
We can all look at two charts and spot relationships between the two. But sometimes you have to put the two charts into one, to make the relationship obvious. You often need a second Y axis for this.

The SystemTemps.csv file also includes fan speed information - let's plot that on the second axis, to see if there's a relationship between CPU fan speed and CPU temperature. Here's how:

```
clichart -fcl 0,1 -t "System Temperatures 17 December" -y "Degrees C" \
        --columnlist2 3 --ytitle2 RPM samples/SystemTemps.csv
```

Note that all options for the second axis use long names, like `--columnlist2`. All options in clichart support long forms, but the most common ones also have a short form, like `-l`.

And here's the result:



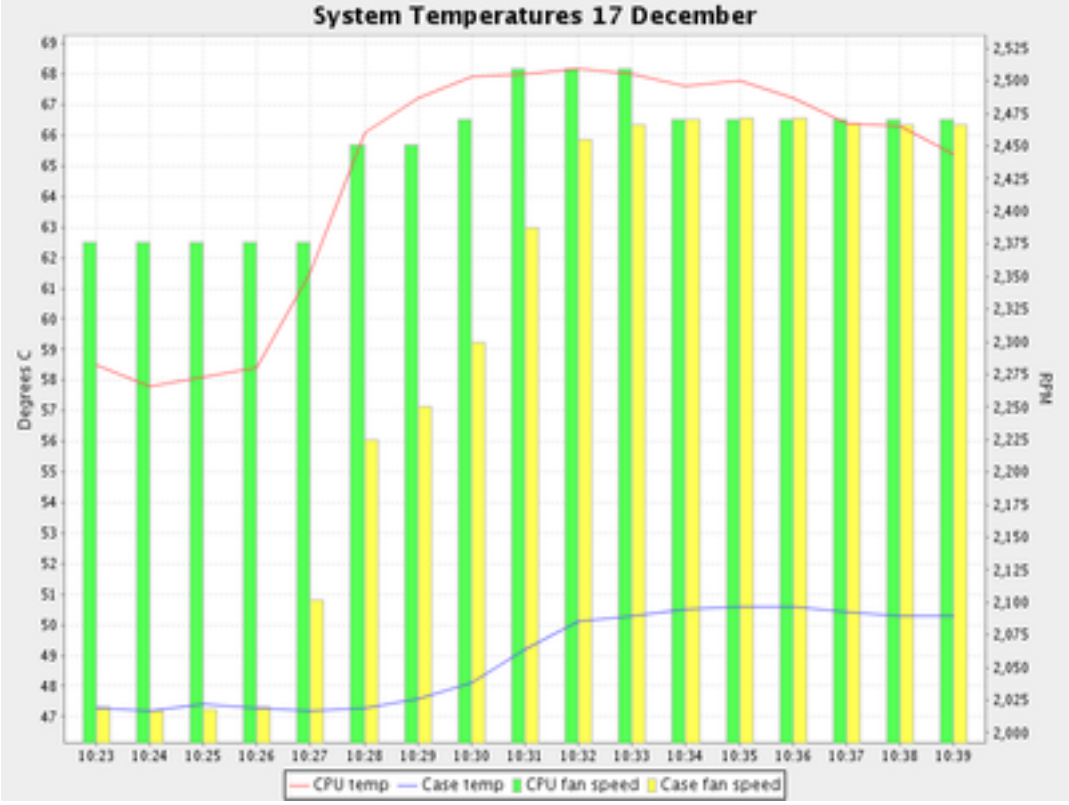
2.3.4 Bar Charts

It's often useful to use a bar chart (a histogram), particularly for discrete data such as counts. clichart allows you to set either Y axis to display as a bar.

Let's turn the second axis in the previous example into a bar, by using the `--bar2` option:

```
clichart -fcl 0,1,2 -t "Temperatures/fan speeds 17 December" -y "Degrees C" \
        --columnlist2 3,4 --ytitle2 RPM --bar2 samples/SystemTemps.csv
```

And the result is:

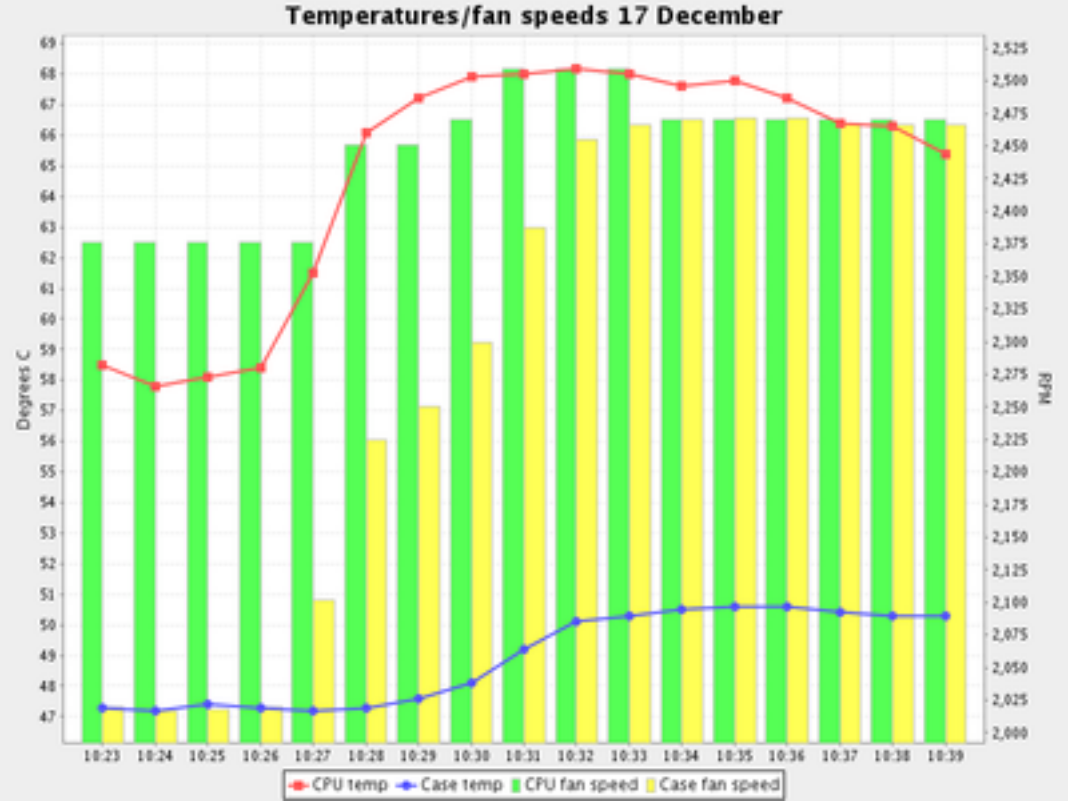


2.3.5 Other Embellishments

You also get some control over the line weight (effectively the line or bar thickness for the chart), and whether data points are displayed with shapes. Here's the previous example, but with data points and thicker lines:

```
clichart -fcl 0,1,2 -t "Temperatures/fan speeds 17 December" -y "Degrees C" \
        --columnlist2 3,4 --ytitle2 RPM --bar2 \
        --datapoints --lineweight 2 --lineweight2 4 samples/SystemTemps.csv
```

And here's the result:



2.4 Charting Patterns in a Log File

In the samples directory, you'll find `System.log`, which contains an extract of messages from a server log. We need to find out from the log file what the transaction rates were over the course of the day, and how memory and thread usage varied.

2.4.1 Plotting Data - Memory and Threads

Let's start with memory or threads. These are an example of data that already exists in the log file - we just need to extract it and put it into a suitable form.

Looking at the log file, you'll see there's a component called `VMStatusLogger` that logs details of memory and thread use every minute, e.g.

```
00:00:45,219 INFO  VMStatusLogger  Memory: 453.27 MB free, 510.43 MB total, Threads: 74 active
```

First, we'll use some UNIX commands to strip out the lines we're interested in and extract the data, then we'll pipe the results to clichart to display

```
grep VMStatusLogger samples/System.log \
    | awk '{print substr($1, 0, 5), $5, $8, $12}' \
    | clichart -l 0,1,2,3
```

And the result is a window showing the following graph (click to see the full-size chart):

Notes:

- We could have used cut instead of awk, or any scripting language (python, perl etc.)
- Instead of using the `substr` function in awk, we could have told clichart to use the full time format, which in this case would have required the option `-d HH:mm:ss,SSS`
- If you're using Windows, Cygwin is your friend - it provides all these tools and more
- The data is separated by whitespace, which is the default for clichart.

CLIChart provides a Python script called linestats, which we could use instead of awk and grep. This is a bit like using a sledgehammer to crack a walnut, but if you don't have access to the UNIX tools (you fool! why not?!), this will do the job. Make sure you have Python installed to use this one.

```
linestats.py -m VMStatusLogger -k s:0:5 -v f:4 -v f:7 -v f:11 \
        -l k,0:min,1:min,2:min samples/System.log \
        | clichart -l 0,1,2,3
```

The result is the same, although the extra power of linestats leads to a more complex command line. Note, however, that linestats has other options that could be useful here, e.g. outputting as CSV, or including a title line in the output (to get a nice legend on the chart). See the linestats documentation for further information.

Well, memory and thread use looks OK - the red line (Series 1) is the amount of free memory inside the system, so we're not about to run out any time soon.

2.4.2 Extracting Rates - Transactions

Let's turn our attention to transactions. In this case, the data in the logs isn't what we want to view - instead, we're interested in seeing the rate at which things happen. In other words, we first need to summarise the data in the logs, then plot the statistics that result.

Each time a transaction is processed, the log contains a line like this:

```
00:00:44,448 INFO  Transaction  A:100 C:0 R:0
```

We're interested in how many transactions the system is processing per minute. What we really need to do is:

- Extract every Transaction line from the log
- Extract the hour and minute timestamp from each line
- For each different timestamp, output the number of times it occurs (which is the number of transactions for that particular minute).

This is where the linestats script comes into its own, although this example only uses a little of its power. One of the things it will do is output counts for each different key (a classifier for grouping lines of data) occurs.

Here's an example of its output:

```
linestats.py -m Transaction -k s:0:5 -l k:cnt,k samples/System.log | head
36        02:30
39        02:31
37        02:32
38        02:33
44        02:34
45        02:35
51        02:36
56        02:37
26        02:38
23        02:39
```

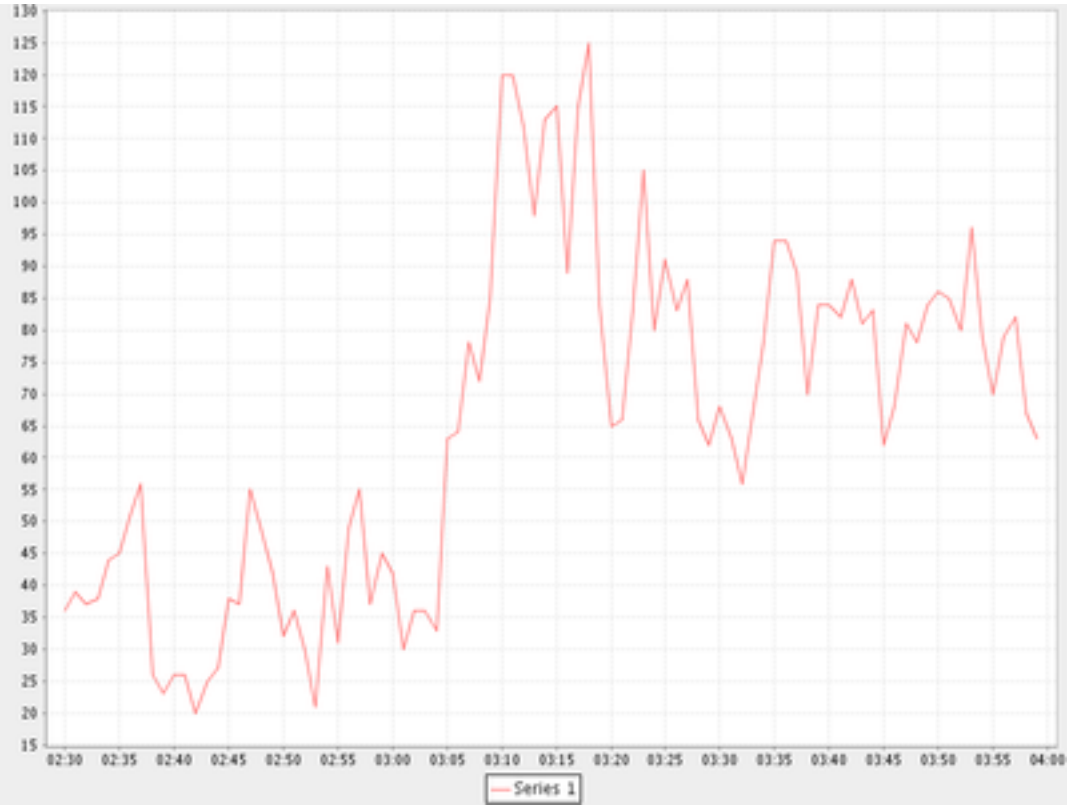
Notes:

- The -m option specifies that we only include lines containing 'Transaction' (this is actually a regular expression, but we didn't need that power here)
- The -k option 's:0:5' specifies that the key for each line is a substring from character 0 (inclusive) to 5 (exclusive), in other words the hours and minutes in the timestamp
- The -l option 'k:cnt,k' specifies the columns we want in the output - in this case, the count for the key, followed by the key itself.

So now we can feed this summary data into clichart to see what the system was really doing:

```
linestats.py -m Transaction -k s:0:5 -l k:cnt,k samples/System.log | clichart -l 1,0
```

And the result is a window showing that the system was pretty busy from around 3:10 am:



As you can see, there's nothing like a chart for showing patterns in data!

We could also make the chart prettier by adding an option to linestats to feed column titles to clichart, and add chart and axis titles to clichart.

2.4.3 Extracting Statistics

The transaction rate in the previous example just touched the surface of the subject of extracting summary data from logs. Very often, there is one or more numeric value on certain lines in the input, and you'd like statistics (minimum, maximum, average, total etc.) on those values.

As an example, consider the Transaction log lines in the previous example. If we assume that the part of the line starting with 'A:' contains the amount of the transaction (in cents, perhaps), we might want to know the minimum, average and maximum transaction amounts for every minute over the course of the day.

By now it will come as no surprise to find that linestats is your friend. But since we're asking it to do more work, its command line is more complex. We've also taken the opportunity to add some column titles to make the clichart output prettier:

```
linestats.py -m Transaction -k s:0:5 -v 'r:A:(\d+)' \
        -c -l k,0:min,0:av,0:max -f 'Timestamp, Min, Average, Max' samples/System.log \
        | clichart -cl 0,1,2,3 -f -y "Transaction amount (cents)"
```

And here's the result:
Notes:

- The -v option specifies the field we want to generate statistics for. Prefixing it with 'r:' makes it a regular expression, and the bracketed part (containing one or more digits) will be extracted as the field value. Note the use of single quotes to protect it from the shell
- The -l option 'k,0:min,0:av,0:max' specifies the output columns as the key itself (i.e. the timestamp), then the minimum, average and maximum values for field number 0 (i.e. the first -v option)
- The -f option provides a first line (with column headings) to add to the output. Note the matching -f option in clichart.

2.4.4 Extracting Statistics for Discrete Values

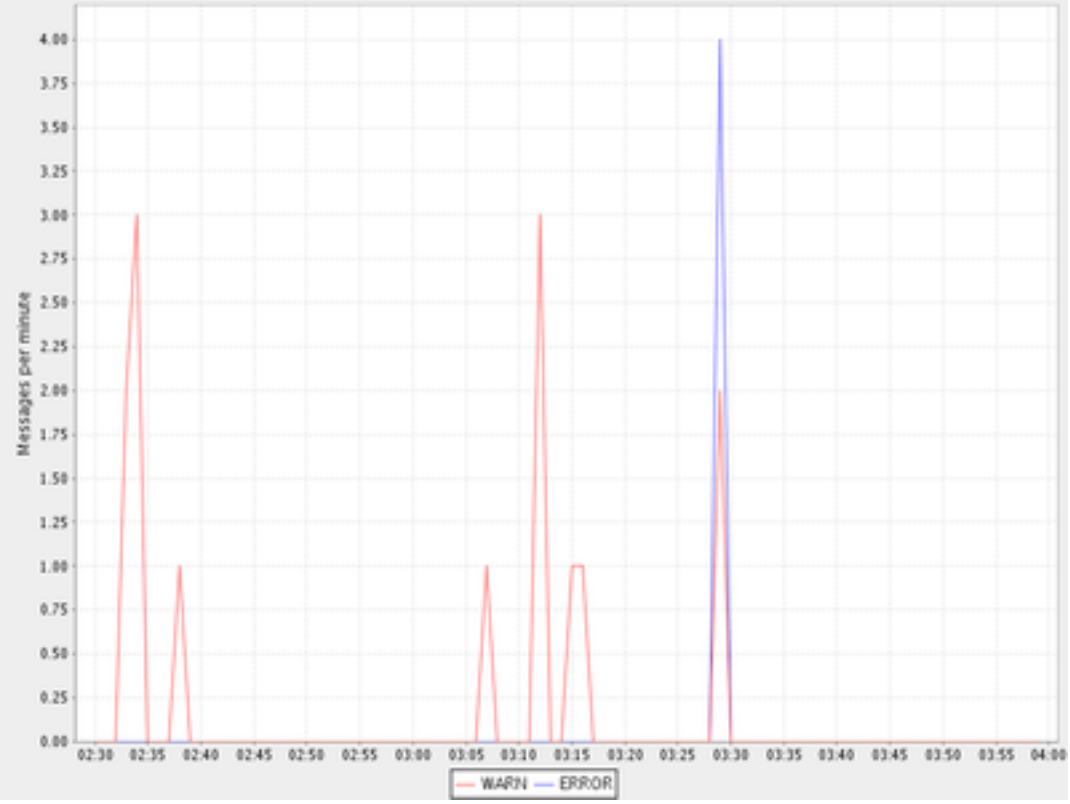
In the previous examples we've looked at the rate at which events happened, and statistics for numeric fields in the log. Often, however, we have a field in the logs that contains discrete values, and we're interested in the rate at which each of these values occurs.

A very common example of this would be to chart the number of ERROR and WARN messages in the log per minute. This gives a quick overview of problems in the system, which you can then drill into. This requires a slightly different technique than previous examples, since we no longer know the columns to be output from the data - instead, we expect to see one column for each discrete value in the field.

This time, instead of linestats we need another program from the CLIChart stable - discretestats. You can probably guess what it's for, from the name.

```
discretestats.py -k s:0:5 -v f:1 -c samples/System.log \
        | clichart -cl 0,2,3 -f -y "Messages per minute"
```

And here's the result:



Notes:

- The -v option specifies the field containing discrete values. Prefixing it with 'f:' interprets this as field number 1
- The -l option to clichart omits column number 1. The output from discretestats has 4 columns: the timestamp key, and 1 column for each discrete value (INFO, WARN and ERROR). We were only interested in the latter 2 of these, so we omitted column 1.

2.4.5 Another Example - Apache Log File

System logs have all sorts of different date/time formats, and clichart allows you to specify the format used in the log (see clichart date formats for details).

Note: This example isn't based on a file in the samples directory.

To show an example of this, we'll use an Apache log, whose standard date/time format looks something like `16/Dec/2006:14:28:03`. If your Apache log format is anything like mine, the date/time is the fourth field (and has a "[" tacked on the front), while the URL requested is the seventh field.

Let's assume that you want to see how many times a minute a set URL is called. This will require:

- Grepping out the URL you're interested in (I'll assume that it's a static URL, so there's no need to worry about dynamic parameters)
- Extracting the day, month, year, hour and minute from the date/time
- Using linestats.py to count the occurrences
- Passing the data to clichart, with the appropriate date format - see clichart date formats.

And here's how it looks on the command line:

```
grep /someurl /var/log/httpd/access_log \
    | awk '{print substr($4, 2, 17)}' \
    | linestats.py \
    | clichart -d "dd/MMM/yyyy:HH:mm"
```

You could do the same using linestats to replace grep and awk (useful if you're using Windows):

```
linestats.py -m /someurl -k "r:\[(\d\d/.../\d\d\d\d:\d\d:\d\d:\d\d)" \
    /var/log/httpd/access_log \
    | clichart -d "dd/MMM/yyyy:HH:mm"
```

Notes:

- The second example uses 2 regular expressions on linestats. The first (the -m option) is to include only lines containing that URL, while the second uses a bracketed section to extract the part of the timestamp we want. See the linestats documentation for details.
- Note the quoting of the -d option to clichart. This is for the benefit of Windows users, since Windows requires quoting of values containing colons.

2.4.6 Aggregating Existing Tabular Data

So, now you've used the above techniques to extract and chart statistics from your log files, and every day you have a new set of statistics files and charts to look at. But you're a busy person, and it takes a while to check all the charts every day. And most days there's not much of interest anyway...

What you need is a way to aggregate data from the day-by-day statistics files, and chart that. For example, let's say you extract statistics on memory and thread usage every day, and save them to CSV files, as in the earlier example. If you had a chart showing some important summary data for all days, you could quickly scan that to see if you need to drill into the detail charts.

Aggregate charts are also really useful for comparing each day with 'normal' (whatever normal is). They make it very easy to spot long-term trends (like memory leaks).

This is where the *aggregate* script comes in handy. Let's say we want to chart the total memory at the end of each day, and the average and maximum thread count during the day. For the purposes of the example, we'll assume that the base data for each day is in a file called *System.log.yyyy-mm-dd.memoryThreads.csv*, where *yyyy-mm-dd* is the log date, and that the columns are the date, memory free, total memory and threads. In the simplest example, we use some unix tools to find the files and work out their dates, then we use *aggregate* to extract a single line of summary data from each day, and append it to a summary file:

```
for f in `find someDir -name "System.log.*.memoryThreads.csv"`; do
    logDate=`echo ${f:11:10}`
    aggregate.py -cf -p $logDate -l 2:last,3:av,3:max $f >> MemoryThreadsSummary.csv
done
```

The summary file will contain a line for each day, looking something like this:

```
...
2007-06-02, 37.9, 64.1, 93
2007-06-03, 47.3, 62.5, 118
...
```

The file doesn't have any column headers, so we'll use the *seriestitles* option in clichart:

```
clichart -cl 0,1,2,3 -d yyyy-MM-dd \
        --seriestitles "Total memory at end of day,Average threads,Max threads" \
        MemoryThreadsSummary.csv
```

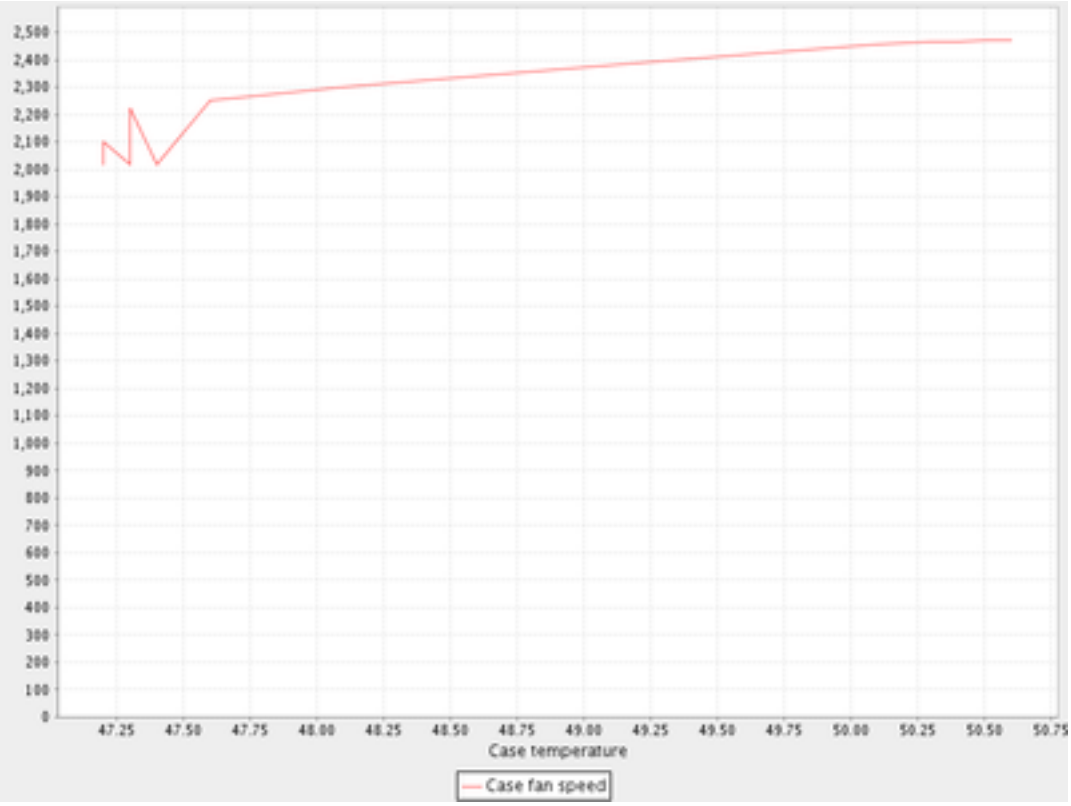
See the aggregate script documentation for more information.

2.5 Charts without Dates/Times

clichart isn't limited to plotting data values against time - you can use values as the X axis, or nothing. For example, to return to our system temperatures, it seems like there might be a relationship between the case temperature and the case fan speed. So let's try plotting the speed against temperature, and find out.

```
clichart -cfvl 2,4 -x "Case temperature" samples/SystemTemps.csv
```

And here's the result:



So it looks as if there is some sort of relationship, but it's not as smooth as we'd hoped (probably because of my dodgy data...).

2.6 CLI Server Mode

clichart can act as a command-line server, for embedding in another program or script. You'd use this mode when you have a number of charts to generate from a script, and you don't want the expense of launching clichart anew for each chart. See the documentation on CLI server mode for details and an example, as well as the cliserverlib documentation for a Python driver for clichart in CLI server mode.

CHAPTER 3

Download and Issues Tracker

3.1 Downloading CLIChart

You can download CLIChart from the releases page on Github.

3.2 Documentation

The home page for CLIChart documentation is at http://clichart.readthedocs.org/en/latest/index.html/.

3.3 Issues Tracker

The CLIChart issue tracker is a good place to ask questions and get answers.

3.4 Source Code

The source code can be found on Github. Pull requests are welcome.

CHAPTER 4

Installing CLIChart

4.1 Introduction

This page is yet to be written properly. So the short form is:

1. Make sure you have Java installed
    - At least Java 1.5 (aka Java 5) is required
    - Java 1.6 (aka Java 6) is recommended
2. Make sure you have Python installed. Any version from 2.4 on should be fine, but the lastest versions tend to be faster (tested with 2.7)
3. Extract the CLIChart zip file to a suitable directory
4. If you want the tools to be installed in your path (recommended):
    - Install easy_install in the setuptools module
    - Use `easy_install clichart-x.y.z.egg` to install (insert your version number)
        - Note that on Linux you'll have to be root (or use sudo)

Test your installation by typing `clichart -h` at a command prompt - this should give you the help screen.

TODO - expand.

CHAPTER 5

clichart

5.1 Introduction

This page documents the **clichart** tool, which is the main tool in the CLIChart project.

Clichart is a Java program (with shell script and batch file wrappers, for UNIX/Linux and Windows respectively). Its job is to generate a chart from tabular data, and either to display the chart in a window or save it to disk.

Clichart can:

- Accept data from stdin or from a file
- Accept input as comma-separated or whitespace-separated
- Produce charts plotted against time (any units down to seconds), numbers, or just as arbitrary values
- Plot any number of data series, on one or two Y axes
- Treat the first row of data as headers for the data series
- Include titles for the chart, X axis and Y axis/axes
- Plot first or second axis as line or bar chart (histogram)
- Control plot line/var thickness, and whether data points display
- Display charts in an interactive window, allowing full customisation of display
- Save charts as JPEG or PNG, either from the command-line or in the display window
- Operate in CLI server mode, generating 1 or more charts based on commands passed via standard in (either from another script, or from a command file)
- Operate in TCP/IP server mode, generating 1 or more charts based on commands passed via a TCP/IP socket (e.g. from cliserverlib).

5.2 Usage

You use the tool like this:

```
clichart [options] [inputFile]
```

5.2.1 Options

```
-b,--bar             Show as a bar chart, not X-Y line
   --bar2            Show second axis as a bar chart, not X-Y line
-c,--csv             Expect input as CSV.  Default is
                     whitespace-separated
   --cliserver       Run the program as a CLI server, reading all
                     commands from standard in
   --colours         Override default chart colours.  Consists of a
                     comma-separated list of 'index:colour', where 'index' is the 0-based
                     series index, and 'colour' is 'red', 'blue', 'green' etc (see
                     documentation).  Indexes not overridden use default colours
   --columnlist2     List of columns for second y axis (if any),
                     comma-separated, 0-based.
-d,--dateformat      Format of date/time, in SimpleDateFormat notation.
                     Defaults to 'HH:mm'
   --datapoints      Indicate each data point
   --datapoints2     Indicate each data point for the second Y axis
-f,--hasheader       First row of data provides column headers for the legend
                     (default is no header row)
   --forceyrange     Force the y axis to use the limits (minimum/maximum) provided.  Default
                     is to use limits only if chart values would exceed them
   --forceyrange2    Force the second y axis (if any) to use the limits (minimum/maximum)
                     provided.  Default is to use limits only if chart values would
                     exceed them
-g,--height          Chart height in pixels (defaults to 600)
-h,--help            Show usage (this screen) and exit
-i,--ignoremissing   Ignore missing columns (default is to terminate)
   --ignoreempty     Ignore empty columns (default is to terminate)
-l,--columnlist      List of columns, comma-separated, 0-based.  X axis
                     value (if any) must be first.  Defaults to '0,1'
   --lineweight      Line weight (values are 1 - 5)
   --lineweight2     Line weight for the second Y axis (values are 1 - 5)
-m,--maxy            Maximum value for y axis
   --maxy2           Maximum value for second y axis (if any)
   --miny            Minimum value for y axis
   --miny2           Minimum value for second y axis (if any)
-n,--noxvalue        Chart has no X axis values - just number the rows
                     instead
-o,--outputpath      Output chart to the given path (otherwise shows in a
                     window), must be JPG or PNG
-p,--ignoredup       Ignore duplicate X axis values (default is to
                     terminate)
   --port            Port on which server should listen (only if TCP/IP
                     server required)
   --seriestitles    Data series titles, comma-separated. Interpreted in
                     same order as Y axis values in the column list
   --seriestitles2   Second axis data series titles, comma-separated.
                     Interpreted in same order as second axis column list
-t,--title           Title for the chart
-v,--xvalue          Chart has simple values as the X axis, not dates or
                     times
-w,--width           Chart width in pixels (defaults to 800)
-x,--xtitle          Title for the X axis
-y,--ytitle          Title for the Y axis
   --ytitle2         Title for the second Y axis (if any)
```

If no input file is provided, the tabular data is read from standard input.

5.2.2 Date Formats

The data format (-d option) uses format strings specified by Java's SimpleDateFormat (see the specification table here).

The most common format string elements are:

Format	Meaning
yy	Year, 2 digits
yyyy	Year, 4 digits
MM	Month (1-12), 1 or 2 digits
MMM	Month abbreviation, e.g. Jan, Feb
dd	Day of the month (1-31), 1 or 2 digits
HH	Hour (0-23), 1 or 2 digits
mm	Minute (0-60), 1 or 2 digits
ss	Second (0-60), 1 or 2 digits
SSS	Millisecond (0-999), 1-3 digits

Other characters (such as ":", ",") are used as-is.

Some common examples:

Format String	Description	Sample Data
HH:mm	Hour (24-hour clock) plus minute	09:32, 23:06
HH:mm:ss	Hour (24-hour clock) plus minute and second	09:32:12, 23:06:01
dd/MMM/yyyy:HH:mm	Apache log format (without second)	07/Apr/2006:09:32
dd/MM/yy HH:mm	Short date with 2-digit year, plus hour and minute. You must enclose this in double quotes (") on the command line because of the space in the format string.	07/04/06 09:32

5.2.3 Colours

The colour override format (–colours option) allows a list of series indexes (0-based) and colours to be specified - any series index not overridden will use the standard clichart colours. Note that series indexes continue from the first axis to the second, so if you have 2 series on the first axis and one on the second, colour indexes 0 and 1 will refer to the first axis, and index 2 to the second.

Colours can be specified in one of 2 ways:

- As a 3-byte hexadecimal number (case-insensitive) specifying the red, green and blue components, e.g. `ff0000` for red. This is the same as the system used in HTML
- By name, chosen from the following list (case-insensitive):
  - black
  - blue
  - cyan
  - darkgrey or darkgray
  - grey or gray
  - green
  - lightgrey or lightgray
  - magenta

  - orange
  - pink
  - red
  - white
  - yellow

For example, to override the first and third series colours, you could use:

```
0:blue,2:ff00ff
```

5.3 Notes

- Requirements to run clichart:
  - Clichart requires Java - see the installation page
  - The clichart script requires that either:
    - You have the `JAVA_HOME` environment variable set correctly - the Java executable must be found at `$JAVA_HOME/bin/java` (Linux/UNIX) or `%JAVA_HOME%\bin\java.exe` (Windows), or
    - The Java executable (`java` for Linux/UNIX or `java.exe` for Windows) is in you PATH.
- Clichart can be used in 3 main modes
  - Interactive, where the chart is displayed in a window (the default), and
    - Right-click on the chart to get the popup context menu
    - The chart can be saved using the File | Save as menu item, pressing Control-S, or from the popup menu
    - Most aspects of the chart's display can be controlled by using the Properties menu item on the popup menu. However, note that control of a second Y axis is not yet supported
    - The popup menu also allows printing and zooming
    - The window can be closed using the File | Exit as menu item, or pressing Control-Q
  - Automatic, where you provide a filename for saving the chart (using the -o option). Clichart will exit after the chart is generated
  - CLI server, where commands are passed via standard input, either from a script or a command file. See the CLI Server Mode section below.
- Clichart is usually executed using the `clichart` wrapper script. The examples assume that you have used easy_install to install CLIChart, in which case the wrapper script is in your PATH. However, clichart can also be executed directly, by replacing `clichart` with `java -jar clichart-0.5.0.jar` (assuming you're using version 0.5.0).
- PNG and JPEG image formats are supported for saving of charts, and these are determined based on the file extension, which must be .png, .jpg or .jpeg (case-insensitive). PNG is recommended, as the image files are smaller, and the images are clearer
- Arguments containing spaces must be quoted, e.g. with double quotes. This is commonly required when setting chart or axis titles
- On Windows, arguments containing colons must be quoted with double quotes. This is commonly required for the -d option, e.g. `-d "HH:mm:ss"`

- Series titles for either axis are comma-separated, so the titles themselves cannot contain commas
- If any two data points have the same X axis value, generation of the chart will fail (TODO: insert error message). Timestamps are evaluated to the second, so timestamps must be at least 1 second apart. Alternatively, use the `-p` option to ignore duplicate values

5.4 Examples

See the quick start guide for examples of using this tool.

5.5 CLI Server Mode

If run in CLI server mode (with the `--cliserver` option), clichart reads commands from standard in, and responds to each successful command by writing a line starting with 'OK' to standard out. This allows another program, script or batch file to drive clichart to produce any number of charts, without the expense of launching clichart anew for each one. To use this mode, you must already have the tabular data available in files.

Each command consists of a command name and an optional argument, followed by a line ending (LF or CRLF). The command name is generally one of the options supported by clichart, either the short or long form, with the following exceptions noted below. Arguments for commands follow the same rules as for the options, except that everything after the command name to the end of the line is treated as the argument, so you shouldn't use quotes around multi-word arguments.

Differences between clichart options and CLI server commands are as follows:

- The path to the input file for the next chart is specified using the `inputPath` command, which takes the path to the (tabular data) input file as its argument. This is required
- The `outputPath` command is also required
- The `go` command forces generation of a chart using the current options
- Options are retained after generating a chart, so generating the next chart only requires changing the options that should change. The `clear` command resets the options to their defaults. **Note:** A number of clichart options do not have any way to reset them to their defaults other than using `clear`.
- Terminate the session by using the `quit` command, closing the calling program (e.g. using Ctrl-C), or closing the standard input stream
- For debugging purposes, use the `debug-echo` command, which echos all commands received to standard error
- Command names are not case-sensitive
- Blank lines and lines starting with # are ignored
- The following options cannot be used as commands: `clichart`, `h` and `help`
- The `timeout` command sets a timeout (in seconds). If the server does not get any commands within this time period, it will exit. This is intended to make the server mode more robust when used in long-running processes.

5.5.1 How to use CLI Server Mode

There are several different options for using CLI server mode. These include:

Saved command file Probably the simplest mode of operation. Save all the required commands to a file, then pipe or redirect that file to clichart, e.g.

```
$ clichart --cliserver < someCommands.txt
```

Shell scripts/batch files Write a shell script or batch file that generates the commands to be run, and pipe the output to clichart. In this case, all output from clichart will appear on standard out, e.g.

```
$ head myscript.sh
#!/bin/sh
INPUT_DIR=some/dir
echo "inputFile $INPUT_DIR/someData.csv"
echo "outputFile someData.png"
echo go
...
# myscript.sh | clichart --cliserver
OK
OK
OK
OK
```

Use cliserverlib to embed in a Python script The cliserverlib.py Python library provides a library to drive clichart in CLI server mode from a Python script.

Embed in a script or program in another language It's easy to write a driver for the CLI server mode in any other language, and to use that in your scripts or programs. The cliserverlib.py Python library provides a useful example of how to do so.

Interactive via console You can also drive clichart via the console, which is useful for testing and debugging. Start clichart using `clichart --cliserver` and type in every command line. After each line hit Enter, and you should see clichart respond with a line starting with 'OK'.

5.5.2 Example CLI Server Session

Here's a sample transcript of a CLI server session. You could run this in any of the ways listed above, but the example shows interacting via the console. Note that `$` is the shell prompt in the example below, and all the `OK` lines are output by clichart, not entered by you. There is no prompt while interacting with the CLI server.

```
$ bin/clichart --cliserver
OK
inputpath samples/SystemTemps.csv
OK
outputpath /home/johnd/tmp/SystemTemps1.png
OK
csv
OK
hasheader
OK
title This is a chart of system temperatures
OK
go
OK
quit
$
```

CHAPTER 6

cliserverlib

6.1 Introduction

This page documents the **cliserverlib** Python library, which provides a Python driver for clichart in CLI server and TCP/IP server modes.

6.2 Requirements

Before you use cliserverlib in CLI server mode, you must ensure that cliserverlib can locate the clichart shell script/batch file (as appropriate). There are 3 ways of doing this:

- The directory containing the `cliserverlib.py` file also contains the clichart shell script/batch file (as appropriate)
- (Unix/Linux only) The directory containing the `cliserverlib.py` file contains a symbolic link to the clichart shell script, or
- The directory containing the clichart shell script/batch file is in the PATH.

To use cliserverlib in TCP/IP server mode, the server must be listening to the appropriate port on localhost.

You'll also need to add the directory containing the `cliserverlib.py` file to your PYTHONPATH, so that it can be found by your Python script.

6.3 Class Documentation

To interact with the library, you use the `ClichartDriver` class, which has the following public methods:

class ClichartDriver(responseTimeout=10, port=-1) Creates the driver, including starting clichart and connecting to its input, output and error streams. `responseTimeout` is the maximum time to wait for any command response before generating an error, in seconds. If `port` is greater than 0, attempts to connect to the CLIChart TCP/IP server on that port

generateChart(clearFirst = True, **kw) Generate a chart, using all the options set using key:value pair arguments. The arguments are as shown in the table below.

By default this clears all previous options first, so you must supply all options required. However, by passing `clearFirst = False`, your options will be taken as overrides for the previously-supplied options.

Any errors will be thrown as `ClichartErrors`.

close() Shut down clichart

setServerTimeout(timeInSeconds) Tell the server to exit if it goes for more than this number of seconds without receiving any input. Used to make use of the server more robust for long-running processes

The module also defines the following values, which are used for the `chartType` argument for the `generateChart()` method.

CHART_TYPE_DATETIME Define for charts with an X axis based on date and/or time (the default)

CHART_TYPE_VALUE Define for charts with an X axis based on numerical values

CHART_TYPE_NONE Define for charts with an X axis with no values (the chart is just a series of Y axis values)

6.3.1 Arguments for generateChart()

The arguments used for the generateChart() method mostly match the documented options for clichart - see the documentation on CLI server mode for documentation on the meaning of each one.

The arguments are:

Argument	Value	Equivalent clichart option	Example
inputPath	Path to input file	inputPath	inputPath = '/path/to/f
outputPath	Path to output file	outputpath	outputPath = 'dir/char
columnList	List or tuple of column indexes	columnlist	columnList = [0,1,4,5
columnList2	As per columnList	columnlist2	columnList2 = [3]
isCsv	True	csv	isCsv = True
hasHeader	True	hasheader	hasHeader = True
chartType	CHART_TYPE_VALUE, CHART_TYPE_NONE	xvalue, noxvalue	chartType = cliserverli
title	Title	title	title = 'The title'
xTitle	Title	xtitle	xTitle = 'Date'
yTitle	Title	ytitle	yTitle = 'Temperature
yTitle2	Title	ytitle2	yTitle2 = 'RPM'
ignoreMissingColumns	True	ignoremissing	ignoreMissingColumn
ignoreDuplicateValues	True	ignoredup	ignoreDuplicateValue
dateFormat	Format string	dateformat	dateFormat = 'HH:MM
showDatapoints	True	datapoints	showDatapoints = Tru
showDatapoints2	True	datapoints2	showDatapoints2 = Tr
lineWeight	int, 1 - 5	lineweight	lineWeight = 4
lineWeight2	int, 1 - 5	lineweight2	lineWeight2= 1
maxY	float	maxy	maxY = 30000
maxY2	float	maxy2	maxY2 = 30000
minY	float	miny	minY = 30000
minY2	float	miny2	minY2 = 30000
forceYRange	True	forceyrange	forceYRange = True
forceYRange2	True	forceyrange2	forceYRange2 = True
height	int, pixels	height	height = 600
width	int, pixels	width	width = 800
isBar	True	bar	isBar = True
isBar2	True	bar2	isBar2 = True
seriesTitles	List or tuple of series titles	seriestitles	seriesTitles = ['Fan sp
seriesTitles2	As per seriesTitles	seriestitles2	seriesTitles2 = ['Fan s
debugEcho	True	debug-echo	debugEcho = True
colours	List of (int, string) as (index, colour)	colours	colours = [(0: 'cyan'),

6.4 Usage and Example

A sample script fragment to generate 2 charts, then exit:

```
# import the library
import cliserverlib

try:

    # create the driver
    driver = cliserverlib.ClichartDriver()

    # a simple chart
    driver.generateChart(title = 'A title', inputPath = 'samples/SystemTemps.csv',
        isCsv = True, hasHeader = True, outputPath = 'samples/SystemTemps1.png')

    # re-use most of the previous options, by setting clearFirst = False
    driver.generateChart(clearFirst = False, outputPath = 'samples/SystemTemps2.png',
        columnList = [0, 1, 2])

    # generate a value-based chart
    driver.generateChart(clearFirst = False, outputPath = 'samples/SystemTemps3.png',
        chartType = cliserverlib.CHART_TYPE_VALUE, columnList = [1, 2])

    # now shut the driver down
    driver.close()

except cliserverlib.ClichartError, message:
    print 'Chart generation failed with message', message
```

CHAPTER 7

linestats

7.1 Introduction

Linestats is a Python script for generating summary statistics from lines of textual data, such as a system log file. It is intended to be used to extract summary data from input, which is then piped to clichart for graphical display.

Use linestats when you have more lines in the input than you need in the output data. Linestats provides various ways to summarise data that share the same 'key field'. All summary values (count, minimum, average, maximum etc.) are grouped based on the value of that key field.

When parsing system logs, the most common use for linestats is to summarise data by time. In this case the key field is part or all of the timestamp in each log message - in this scenario, linestats will output one line of statistics (whichever statistics you choose) for each unique timestamp.

Linestats can:

- Accept data from stdin or from a file
- Identify the key field based on whitespace-separated fields, a substring, or a regular expression
- Accumulate statistics for zero or more value fields
- Identify value fields based on whitespace-separated fields, a substring, or a regular expression
- Output counts for each value in the key field, e.g. for every unique timestamp
- Output minimum, average, maximum and/or total values for each value field, for each value in the key field, e.g. for every unique timestamp
- Ignore input lines that do not contain a supplied regular expression
- Sort the output by the key field
- Output as comma-separated or whitespace-separated
- Include a supplied column heading line in the output, for generating legends in clichart.

7.2 Usage

You use the tool like this:

```
linestats.py [inputOptions] [outputOptions] [inputFile]
```

If no input file is specified, reads from stdin. Output is always to stdout.

7.2.1 Input Options:

```
-h           Show help (this screen) and exit
-k <keyspec> Specifies how to extract a key from each line (default is to use
             the whole line).  keyspec can be:
   s:<substring>  Extract key as a substring.  See Substrings
   f:<index>      Extract key as a field.  Fields are separated by white
                  space, 0-based
   r:<regex>      Extract key as a regular expression
-m <regex>   Only include lines matching this regular expression
-v <valuespec> Specifies a value for which to accumulate statistics from each
             line.  Zero or more.  valuespec must return a numeric value, and
             may be:
   s:<substring>  Extract value as a substring.  See Substrings
   f:<index>      Extract value as a field.  Fields are separated by white
                  space, 0-based
   r:<regex>      Extract value as a regular expression
```

7.2.2 Output Options:

```
-c           Output as CSV (default is whitespace-separated)
-f <line>    Output this line first, as the headers for the columns
-l <columns> A comma-separated ordered list of columns to include in the
             output (default is 'k,k:cnt').  The field index is the 0-based
             number of the field in the list of -v options, i.e. field number
             0 is the field specified by the first -v option.  Columns may be:
   k         The key
   k:cnt     The count for the key
   0:av      The average of field '0'
   0:min     The minimum of field '0'
   0:max     The maximum of field '0'
   0:tot     The total of field '0'
-s           Sort output by key column
```

7.2.3 Other Options:

```
--nojit      Disable Psyco Just-In-Time compiler
```

7.2.4 Substrings:

Substrings are specified by one or two indexes into the line.

- Substring indexes must be separated by a colon (:)
- The substring is taken from the first index (inclusive) to the second index (exclusive)
- Each index is 0-based (i.e. numbers from 0)
- Negative indexes count from the end of the line, i.e. -1 refers to the last character in the line
- If no second index is given, the substring is taken from the first index to the end of the line.

Examples (with 's:' prefix):

```
s:0:5     Extract the first 5 characters from each line
s:32:     Extract from the 33rd character to the end of the line
s:10:-8   Extract from the 9th character to the 9th-last
```

7.2.5 Regular Expressions:

Regular expressions are mainly used to extract key or value fields from lines, although they are also used for the -m (match) option.

Regular expressions follow the Perl 5 syntax as implemented by Python (NOT grep/egrep!). The main difference is that the * and + operators are greedy by default - if you want the egrep behaviour, append *?* to them, e.g. change *ab.*yz* to *ab.*?yz*. See the Python regular expression documentation for full information.

When the regex is used to extract a value, if it contains a bracketed group the value returned for the group is used - otherwise, the entire match is used. E.g. `thread count:  ([0-9]+)` will return the number matched by the bracketed group, while `thread count:  [0-9]+` will return the entire string that matches.

Note that you must quote or escape special characters to prevent the shell from interpreting them, typically with single quotes.

Examples (with 'r:' prefix):

```
'r:^\d\d:\d\d'  Extract the first 5 characters, which must be in the form 99:99
'r:A:(\d+)'     Find the string 'A:' followed by 1 or more digits, and return
                the digits
```

7.3 Notes:

- Linestats requires Python - see the installation page

7.4 Examples

See the quick start guide for examples of using this tool.

CHAPTER 8

discretestats

8.1 Introduction

Discretestats is a Python script for generating summary statistics from lines of textual data containing a field with discrete values, such as a system log file. It is intended to by used to extract summary data from input, which is then piped to clichart for graphical display.

Discretestats is based on the idea of extracting a 'key' field and a 'value' field from each line of data. The most common key field is a timestamp, while the value field can be anything that has discrete values. The output can be thought of as a table - one row for each key field value, and one column for each value field value, with each cell containing the count that value and key. A typical example is generating statistics on the number of error and warning messages in a system log per minute.

The differences between discretestats and linestats are:

- Any value fields in linestats must be numbers (since you're interested in averages, minima etc). The field in discretestats can be anything (but it's usually a string)
- Discretestats tells you how often each **distinct value** occurs for each key, whereas using a count in linestats just shows how many times the key occurs
- In linestats, you specify exactly the columns to be output. In discretestats you don't always know how many columns there will be, since there's a column for each discrete value.

Discretestats can:

- Accept data from stdin or from a file
- Identify the key field based on whitespace-separated fields, a substring, or a regular expression
- Accumulate counts for each discrete value in a single value field
- Identify the value field based on whitespace-separated fields, a substring, or a regular expression
- Output counts for discrete value in thevalue field, for each value in the key field, e.g. for every unique timestamp
- Ignore input lines that do not contain a supplied regular expression
- Sort the output by the key field
- Output as comma-separated or whitespace-separated
- Quote column headings and keys containing spaces and/or commas.

8.2 Usage

You use the tool like this:

```
discretestats.py [inputOptions] [outputOptions] [inputFile]
```

If no input file is specified, reads from stdin. Output is always to stdout.

8.2.1 Input Options:

```
-h           Show help (this screen) and exit
-k <keyspec> Specifies how to extract a key from each line (default is to use
             the whole line).  keyspec can be:
   s:<substring>  Extract key as a substring.  See Substrings
   f:<index>      Extract key as a field.  Fields are separated by white
                  space, 0-based
   r:<regex>      Extract key as a regular expression
-m <regex>   Only include lines matching this regular expression
-v <valuespec> Required.  Specifies a value for which to accumulate statistics
             from each line.  valuespec may be substring, field or regex
             as for -k option.
```

8.2.2 Output Options:

```
-c           Output as CSV (default is whitespace-separated)
-s           Sort output by key column
```

8.2.3 Other Options:

```
--nojit      Disable Psyco Just-In-Time compiler
```

8.2.4 Substrings:

Substrings are specified by one or two indexes into the line.

- Substring indexes must be separated by a colon (:)
- The substring is taken from the first index (inclusive) to the second index (exclusive)
- Each is index is 0-based (i.e. numbers from 0)
- Negative indexes count from the end of the line, i.e. -1 refers to the last character in the line
- If no second index is given, the substring is taken from the first index to the end of the line.

Examples (with 's:' prefix):

```
s:0:5     Extract the first 5 characters from each line
s:32:     Extract from the 33rd character to the end of the line
s:10:-8   Extract from the 9th character to the 9th-last
```

8.2.5 Regular Expressions:

Regular expressions are mainly used to extract key or value fields from lines, although they are also used for the -m (match) option.

Regular expressions follow the Perl 5 syntax as implemented by Python (NOT grep/egrep!). The main difference is that the * and + operators are greedy by default - if you want the egrep behaviour, append *?* to them, e.g. change *ab.*yz* to *ab.*?yz*. See the Python regular expression documentation for full information.

When the regex is used to extract a value, if it contains a bracketed group the value returned for the group is used - otherwise, the entire match is used. E.g. `thread count:  ([0-9]+)` will return the number matched by the bracketed group, while `thread count:  [0-9]+` will return the entire string that matches.

Note that you must quote or escape special characters to prevent the shell from interpreting them, typically with single quotes.

Examples (with 'r:' prefix):

```
'r:^\d\d:\d\d'  Extract the first 5 characters, which must be in the form 99:99
'r:A:(\d+)'     Find the string 'A:' followed by 1 or more digits, and return
                the digits
```

8.3 Notes:

- Discretestats requires Python - see the installation page

8.4 Examples

See the quick start guide for examples of using this tool.

CHAPTER 9

mid

9.1 Introduction

Mid is the steroidally-enhanced child of the venerable Unix head and tail utilities. It's a Python script for extracting ranges of lines from a file (or stdin) in the most convenient way possible.

You can use mid to strip out lines from text input prior to passing it to linestats or discretestats, and/or to strip out lines from tabular data prior to passing it to clichart. The most common uses are to process only data from a selected time period in a system log, or to remove lines of tabular data that cause problems for clichart.

Mid can:

- Accept data from stdin or from a file
- Extract one or more line ranges from the input
- Accept line indexes as positive numbers (counting from the start of the file), and as negative numbers (counting backwards from the end of the file).

9.2 Usage

You use the tool like this:

```
mid.py [options] range [range *] [inputFile]
```

If no input file is specified, reads from stdin. Output is always to stdout.

9.2.1 Options:

```
-h          Show help (this screen) and exit
```

9.2.2 Ranges:

At least one line range is required. Ranges take the form:

```
<startLine>:<endLine>
```

Line numbers are 1-based, i.e. the first line in the file is line 1, and both the start and end line numbers are **inclusive**, i.e. both the start and end lines are included in the output. Negative line numbers count backwards from the end of the file, with -1 being the last line.

9.2.3 Examples:

```
10:20          All lines from 10 to 20
2:-1           All but the first line of the file
-10:-1         The last 10 lines of the file
1:1 -1:-1      Just the first and last line of the file
```

9.3 Notes:

- Mid requires Python - see the installation page

CHAPTER 10

merge

10.1 Introduction

Merge is a Python script for merging tabular data from 2 or more files, based on common key values. Its most common use is when you have data from several sources which you would like to merge in order to display on the same chart.

All merging is based on the values in the key column of each file - these values are frequently dates or times for time-based data. The output will consist of 1 row for each distinct key value found in any of the files, with 1 column for the key plus any columns you specified from each of the files. Column cells for which no value is available, i.e. where the key value was not found in all files, default to an empty string, although this can be controlled using the `--defaults` option.

Note that keys are treated as text, so for instance '7' and '7.0' are treated as different.

Merge can:

- Merge data from 2 or more files
- Accept data in CSV or whitespace-separated formats. Output is in the same format as the input
- Treat the first row in each file as a header
- Employ user-specified default values for any column.

10.2 Usage

You use the tool like this:

```
merge.py [options] inputFile*
```

Output is always to stdout.

10.2.1 Options:

```
-c              Data is CSV (default is whitespace-separated)
-f              First row of file is a header row, and should be skipped
-h              Show help (this information) and exit
-l <columns>    A comma-separated list of fileNumber:columnNumber to be
                included in the output, for all non-key columns (the key column
                will be output at the start of each line).  Both file and
                column numbers are 0-based, e.g. "0:1,0:2,1:1,2:4"
```

```
-k <columns>    A comma-separated list of fileNumber:columnNumber specifying
                the key column for each file.  There must be 1 entry per file,
                e.g. "0:0,1:0,2:1"
--defaults      A comma-separated list of the default values for all non-key
                columns, in the same order as the -l option list.  If omitted,
                values default to the empty string
--nojit         Disable Psyco Just-In-Time compiler
```

10.3 Notes:

- Merge requires Python - see the installation page

10.4 Example:

Merges two CSV files, where the key is the first column in each file. Output the second and third columns from each file.

merge.py -k 0:0,1:0 -l 0:1,0:2,1:1,1:2 -c file1.csv file2.csv

CHAPTER 11

aggregate

11.1 Introduction

Aggregate is a Python script for extracting aggregate (or summary) data from numeric columns in tabular data, such as CSV files.

A typical use for Aggregate is generating data to show long-term trends, where you already have a number of tabular data files covering shorter periods. For example, we use Aggregate to extract day-by-day transaction rates and total volumes, memory and thread use etc. from daily stats for a number of servers - this allows a quick glance to pick up any anomaly that should be investigated further.

Aggregate can:

- Accept data from stdin or from one or more files
- Accept and output comma-separated or whitespace-separated data
- Output minimum, maximum, average, total, count, standard deviation, first or last value for any numeric column
- Ignore the first line of the file, e.g. when it contains column headings
- Output one aggregate row per file, or one per key/group value.

11.2 Usage

The output of the script will be 1 row of data for each input file (or only one row if stdin is used). Alternatively, if a key (aka group) is specified, there will be one row per key. Output is in the same format as the input, i.e. whitespace or comma-separated.

You use the tool like this:

```
aggregate.py [options] [inputFile*]
```

If no input file is specified, reads from stdin. Output is always to stdout.

11.2.1 Options:

```
-c           Data is CSV (default is whitespace-separated)
-f           First row of file is a header row, and should be skipped
-h           Show help (this information) and exit
-l <columns> A comma-separated ordered list of columns and aggregate types
```

```
              to include in the output (required).  See below
-p            A prefix column value to write in the aggregate columns for each
              line.  Pass 1 or more of these, e.g. to provide a date column
              value.  Example: -p 12/03/2007
-s            If no rows of data were found, silently output nothing
-x            A suffix column value to write in the aggregate columns for each
              line.  Pass 1 or more of these, e.g. to provide a date column
              value.  Example: -x 12/03/2007
--nojit       Disable Psyco Just-In-Time compiler.
```

11.2.2 Column indexes

The column index is the 0-based number of the column in the file, while the aggregate type is 'min', 'av', 'max', 'tot', 'cnt', 'sd' (standard deviation), 'first' or 'last'. If the type is 'k', this column is treated as a key, and output is grouped by the values in the column. Note that the count is just the number of lines in the file, not counting any header row. E.g.:

```
1:max,2:av,2:max,0:cnt,4:tot,0:sd
```

Negative column indexes can be used, with -1 representing the last column etc. E.g.:

```
1:max,-1:tot
```

Simple expressions can also be used. To indicate negation rather than a negative column index, you must separate the minus sign and the column index with a space. E.g.:

```
'1:max / 3:cnt / 60', '1:tot - 3:tot', '2:av - -1:av'
```

11.3 Notes:

- Aggregate requires Python - see the installation page
- Every column specified with the `-l` option **must** be numeric
  - The only exception to this is columns specified as keys - these are treated as strings
- Key columns can be specified anywhere in the column list, but will always be output at the start of each row (but in the order specified)
- When key columns are specified, output is sorted in ascending order by the key columns (as strings)
- Expressions can use:
  - Any column aggregate, e.g. '3:av'
  - Integer and floating point numbers
  - Standard mathematical operators: '+', '-', '*', '/', as well as '%' (remainder after integer division)
  - Standard Python functions (if you know some Python). However, you won't be able to use any function requiring more than one argument, since the comma is used to separate the column expressions.
- Expressions must return a single numericial value.

11.4 Examples

Using data from the samples directory, we can extract the minimum, average and maximum CPU temperature and fan speed:

```
$ aggregate.py -cfl 1:min,1:av,1:max,3:min,3:av,3:max samples/SystemTemps.csv
57.80, 64.73, 68.20, 2376, 2448, 2510
```

If we're only interested in what the values were at the start and end of the day:

```
$ aggregate.py -cfl 1:first,1:last,3:first,3:last samples/SystemTemps.csv
58.50, 65.40, 2376, 2471
```

And if we want the date as the first column in the output:

```
$ aggregate.py -cfl 1:first,1:last,3:first,3:last -p 12/03/2007 samples/SystemTemps.csv
12/03/2007, 58.50, 65.40, 2376, 2471
```

Alternatively to generate aggregates keyed on (grouped by) one of the columns:

```
$ aggregate.py -cfl 3:k,1:min,1:av,1:max samples/SystemTemps.csv
2376, 57.8000, 58.8600, 61.5000
2451, 66.1000, 66.6500, 67.2000
2471, 65.4000, 66.9429, 67.9000
2510, 68, 68.0667, 68.2000
```

CHAPTER 12

histogram

12.1 Introduction

Histogram is a Python script for generating a histogram (or frequency distribution) from numeric columns in tabular data, such as CSV files.

A typical use for Histogram is showing the spread and frequency of server response times for a particular function - the histogram gives a different view on the data than the typical minimum/average/maximum data provided by Linestats.

Histogram can:

- Accept data from stdin or from a file
- Accept and output comma-separated or whitespace-separated data
- Generate a histogram for any numeric column in tabular data
- Construct the histogram with any user-specified number of intervals or interval size
- Include percentages for each interval
- Show histogram and percentage data as individual or cumulative
- Ignore the first line of the file, e.g. when it contains column headings
- Output a headings row if required.

12.2 Usage

The output of the script will be 1 row of data for each of the number of intervals specified, containing the start and end values for the interval, and the number of data points falling within the interval. Output is in the same format as the input, i.e. whitespace or comma-separated.

You use the tool like this:

```
histogram.py [options] [inputFile]
```

If no input file is specified, reads from stdin. Output is always to stdout.

Either the number of intervals (-i) or the interval size (-s) is required.

12.2.1 Options:

```
-c              Data (input and output) is CSV (default is
                whitespace-separated)
-f              First row of file is a header row, and should be skipped
-h              Show help (this information) and exit
-i <intervals>  The number of intervals over which the histogram should be
                generated (i.e. the number of output rows)
-l <column>     The number of the column from which to generate the histogram
                (required).  The column index is the 0-based number of the
                column in the file.
-m              Show cumulative interval values, rather than individual
-p              Add a column showing the percentage for the interval
                (individual or cumulative)
-s <size>       Size of interval (an alternative to -i)
--header        Include a header row in the output
--nojit         Disable Psyco Just-In-Time compiler
```

12.3 Notes:

- Histogram requires Python - see the installation page
- The column specified with the `-l` option **must** be numeric

12.4 Examples

Using data from the samples directory, we can generate a histogram for case fan speed (column 4, counting from 0), and dividing the data into 5 intervals:

```
$ histogram.py -i 5 -l 4 samples/SystemTemps.txt
2017 2108 5
2108 2199 0
2199 2290 2
2290 2381 1
2381 2472 9
```

The CSV version of the data also has a header row, which needs to be skipped. We can also add an output header. This timie we'll do the CPU temperature (column 1), which is a floating point number:

```
$ histogram.py -i 5 -l 1 -cf --header samples/SystemTemps.csv
Interval_Start, Interval_End, Count
57.8000, 59.8800, 4
59.8800, 61.9600, 1
61.9600, 64.0400, 0
64.0400, 66.1200, 2
66.1200, 68.2000, 10
```

To chart this using clichart, we'd do something like this (note the -v flag, since the x axis is based on values rather than timestamps):

```
$ histogram.py -i 5 -l 1 -cf samples/SystemTemps.csv | clichart -l 1,2 -cbv
```

CHAPTER 13

Frequently-Asked Questions

13.1 General

13.1.1 How can I make CLIChart run faster?

a) Install Psyco to speed up Python scripts

Psyco is a Just-In-Time compiler for Python. The CLIChart Python-based tools (linestats, discretestats, mid and aggregate) will use psyco if it's installed, and this should speed up their operation when larger amounts of data are involved.

b) Use a Later Version of Java

The main clichart program runs under Java. Later versions of Java are likely to make it start and/or run faster. Java 1.4 is particularly slow - if you're still on 1.4, you should think very seriously about upgrading.

13.2 Running Under Windows

13.2.1 Why do I get an 'invalid file descriptor' error when piping data to a tool?

If you get a stack trace (complaining about an 'invalid file descriptor') when piping data into one of the Python-based tools (linestats, discretestats, mid or aggregate), this is due to a bug present in some versions of Windows.

The bug presents when you pipe data into a script that was launched based on a file extension, so for example it occurs in the first case below but not the second. Note that you must have `python.exe` in your PATH for the second example to work:

```
rem This may fail, depending on the Windows bug
... | aggregate.py

rem This should always work (if python.exe is in your PATH)
... | python aggregate.py
```

See http://support.microsoft.com/kb/321788 for details of the bug

CHAPTER 14

Developing CLIChart

14.1 CLIChart needs your help!

I've been developing CLIChart for many years now, but for the last couple of years I've been using it less and less. And when I do use it, it already does most of the things I want. Therefore I don't have any intention to do much more (if anything) with the project.

So, if you want CLIChart to continue and/or advance, please get involved!

14.2 Getting the source

Go to the project page on Github. You could just clone the git repo directly, but then you won't be able to submit changes, so better to fork the project and then clone your fork - that way you'll be able to submit pull requests in future.

14.3 Contributing your changes

Once you've made some changes that you think will work (and you've run through at least most of the checks in `/DistChecklist.txt`), please submit a pull request on Github.

In future, if anyone's interested in becoming a long-term maintainer, I'm happy to pass on the reins.

14.4 Project structure

Main parts are:

- root directory - gradle build script (`build.gradle`) and properties file (`settings.gradle` `setup.py` (for setuptools install), plus some text files
- `/bin` - a couple of miscellaneous scripts for building and packaging
- `/resource/docs` - documentation root (uses Sphinx/reST)
- `/resource/samples` - sample files for demonstrating capabilities
- `/src/main/java` - Java source for clichart tool
- `/src/main/python` - source for all tools (including clichart wrapper)

- `/src/test/java`
- `/src/test/python`
- `/build` - build directory.

14.5 Building

IMPORTANT NOTE:

> Clichart is set up to build on *nix systems (Linux, OSX etc.). The build can be made to work on Windows under Cygwin, but you'll probably have quite a bit of work to do if you try to do it on Windows directly...

The quick version:

```
# start in the project's root directory
gradle dist                 # builds and zips everything
gradle installForTest
. build/install-test/virtualenv/bin/activate
bin/testInstall.py
deactivate
```

The longer version is documented in `/DistChecklist.txt`. This will ensure that your development environment is set up OK, and then walk you through all the steps to build, test, and package.

14.5.1 Mandatory Build Tools

git Of course

Java JDK

Gradle Build system

Sphinx Documentation builder. Check for `sphinx-build`

Python setuptools Provides packaging and installer. Check for `easy-install`

14.5.2 Recommended Build Tools

The following tools are used in standard build targets, or are useful for checking results.

Python nosetest Test runner for unit tests

Python coverage Code coverage tool for Python, integrates with/called via nosetest

Python virtualenv Provides virtualised Python environments, used for test installs

ImageMagick convert Used to generate thumbnails of charts for documentation

Linkchecker Useful tool to check links in documentation

14.6 Documentation

Documentation is set up on readthedocs.org, with a webhook from the git repo on Github to automatically rebuild the documentation when published.

Special notes about the documentation:

- You have to update both the version and release numbers in the Sphinx conf.py file for a new release. The gradle build script will fail the build if this isn't done
- The documentation builds initially to `/resource/docs/build` - the files are then copied from there to the `/build` directory
- Because the documentation build uses Sphinx, there's no easy way (that I've found, at least) to have the thumbnail images updated automatically and copied in. Therefore, the gradle build script (re)generates thumbnails if required, in the `/resource/docs/source/_static/images` directory. **This means that running a build may require a git commit afterwards**.