Ping v2 CSV Analysis and Manipulation Toolset

Daniel Weibel

29 March 2015

Introduction

Note: *this is a preliminary documentation, presenting some features of the current state of the toolset. The work is in progress.*

The Ping v2 CSV Analysis and Manipulation Toolset currently consists of 21 shell programs in directory `bin` (see listing in Section Shell Programs), and a file with R functions `lib/toolbox.r`.

The purpose of the toolset is to provide flexible and generic building blocks that can be composed to answer any specific questions about the Ping data. This contrasts with the approach of repeatedly tailoring code and tools from scratch for every specific question (e.g. RTT values on train lines). With the toolset, many different questions about the data can be investigated with very little effort, and the users can concentrate on the actual analysis of the data rather than on technical details of script writing.

The shell programs are used from the shell, and most take as input a CSV file, do some transformation on it (e.g. cleaning), and output the transformed CSV file. The R functions serve a double role. One one hand, they are used by the shell programs and serve as a library to them. On the other hand, they can be directly used by the user, for example, interactively in the R console, or in R scripts.

Compatibility

The toolset has been designed to run out-of-the box on Mac OS X and Linux. So far, it has been tested on Mac OS X and Debian 7. Further compatibility tests (especially on other Linux versions) are to be done in the future.

Installation

The toolset is self-contained, and it is installed by simply copying the root directory to the local machine.

However, there are some dependencies on third-party software that must be installed on the local machine. These dependencies are explained in Section Dependencies.

For convenience, the `bin` directory can be added to the `PATH`[1], for example:

```
echo 'PATH=/Users/dw/Desktop/csv/bin:$PATH' >>~/.bash_profile
```

Tutorial

This is a short step-by-step tutorial that demonstrates the most important of the currently implemented features of the shell programs. Note that you can type `cmd --help` for any command for getting information about what the command does and how it is used.

Download CellIdsService data from a certain IMEI over a certain time period:

[1] Make sure to add the `bin` directory to the beginning of the `PATH` variable to prevent that a command of the toolset is shadowed by an existing shell command. You can always test if any command is shadowed in the `PATH` with `which -a <cmd>`.

```
get -i $(peek -I 29) -f CellIdsService -t 2015-03-01_2015-03-31 >cid.csv
```

Remove rows with invalid cells from the data (check the number of rows before and after with `nr cid.csv`):

```
clean -o cid.csv cid.csv
```

Add columns with the date, time, weekday, and network type to the file:

```
add -dtwn -o cid.csv cid.csv
```

Extract rows with distinct cells from the file and sort the resulting rows by cells:

```
distinct cid.csv | order >cid_distinct.csv
```

Download PingService file from same IMEI over same time period:

```
get -i $(peek -I 29) -f PingService -t 2015-03-01_2015-03-31 >ping.csv
```

From the PingService file, extract the rows where the field "rtt" is greater than 0 (i.e. a ping has actually been done) and the field "mobWifi" is MOBILE (i.e. the ping was done on the mobile network). From these rows extract the columns "ts" (timestamp) and "rtt".

```
rows -c 'rtt > 0 & wifiMob == "MOBILE"' ping.csv | cols -k ts,rtt -o ping.csv
```

Merge the RTT values from the modified PingService file into the the CellIdsService data and save result in a new file:

```
mrgts -f ping.csv -c rtt cid_distinct.csv >cid_with_rtt.csv
```

Show the rows with an RTT value of >150 milliseconds:

```
rows -c '!is.na(rtt) & rtt > 150' cid_with_rtt.csv
```

Open the file in the R console (the `toolbox.r` functions will be available in this R environment):

```
openr cid_with_rtt.csv
```

Download the complete OpenCellID database from today (around 530 MB unzipped):

```
ocid_get ocid.csv
```

Extract the region of Western Switzerland from the OpenCellID database and remove duplicates from it (use http://boundingbox.klokantech.com/ to select a bounding box):

```
ocid_extract -b 5.61,45.62,8.47,47.91 ocid.csv | ocid_clean >ocid_ch.csv
```

Merge locations from the OpenCellID file with the extracted distinct cells from the CellIdsService file:

```
mrgcell -f ocid_ch.csv -c lat,lon cid_distinct.csv >cid_loc.csv
```

Calculate the percentage of cells for which the location has been found:

```
prc <(loc -y cid_loc.csv) <(loc -n cid_loc.csv)
```

Create a Google Earth KML file displaying the locations of the cells for which the location has been found:

```
loc -y cid_loc.csv | kml -n "Nexus 6" -s "March 2015" >cid.kml
```

Conventions and Design Principles

The shell programs have been designed with some consistent features and concepts in mind that should facilitate their understanding and usage.

The first thing to note is the `--help` option which displays a help text explaining the purpose and usage of the command. It is recommended to use this feature very frequently.

Furthermore, the shell programs are designed to support a pipe-based workflow, that is, the chaining of multiple commands so that the output of one command goes directly to the input of the next command via a pipe (|).

This design implies that all commands print only relevant data to stdout (i.e. they are not "chatty"), that the default output is stdout, and that the default location to read input from is stdin.

However, for every program the reading of the input can be switched to a file which has to be specified as the last argument on the command line. Similarly, in many commands the output can be redirected to a file with the -o option. This allows for in-place transformations, that is, the saving of the output of a command back to the input file (note that redirecting the output of a command to the input file with > would empty the input file before it is read).

Shell Programs

Current state (29 March 2015).

Program	Purpose
add	Add columns
b	Print body
clean	Remove rows with invalid cells
cols	Extract columns
distinct	Extract the rows with distinct cells
get	Download Ping v2 CSV files
h	Print header
kml	Create KML file with point placemarks for viewing in Google Earth
loc	Extract rows with or without a location
mrgcell	Merge two CSV files by their cell fields
mrgts	Merge two CSV files by their timestamps
nc	Print number of columns
nr	Print number of rows
ocid_clean	Remove rows with duplicated cells from an OpenCellID file
ocid_extract	Extract rows from an OpenCellID file
ocid_get	Download the complete OpenCellID database
openr	Open a file in R for further analysis
order	Order rows by cells
peek	Print available IMEIs and data files of Ping v2
prc	Calculate relation of sizes of two files in percent
rows	Extract rows

Dependencies

The toolset requires that certain non-standard software is installed on the system. In the following, we list these dependencies and give for each one installation instructions for Mac OS X and Linux.

Note that our proposed way of installing software is APT for Linux and Homebrew for Mac OS X. While APT is installed by default in many Linux versions, Homebrew is not installed on Mac OS X, but can be installed very easily as explained at the end of this section.

R

The backbone tool for the whole toolset.

Test if installed

```
R --version
```

Install on Linux

```
sudo apt-get install r-base
```

Install on Mac OS X

```
brew install r
```

gawk

GNU awk. Used to handle quoted CSV fields. Note that the standard awk is not sufficient, it must be gawk.

Test if installed

```
gawk --version
```

Install on Linux

```
sudo apt-get install gawk
```

Install on Mac OS X

```
brew install gawk
```

curl

Used to download files over HTTP.

Test if installed

```
curl --version
```

Install on Linux

```
sudo apt-get install curl
```

Install on Mac OS X

```
brew install curl
```

Install Homebrew on Mac OS X

Homebrew is an easy-to-use package manager for Mac OS X, see http://brew.sh for more information. It can be installed simply with:

```
ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```