

## Status

The former repository of ChatScript was at <https://github.com/bwilcox-1234/ChatScript>, however, I lost access to this repository (2 factor authentication lost that I didn't want in the first place and I could not convince github to restore my access). So it is no longer maintained. But the global user ChatScript became available, and is a better name anyway.

## ChatScript

Natural Language tool/dialog manager

ChatScript is the next generation chatbot engine that has won the Loebner's 4 times and is the basis for natural language company for a variety of tech startups.

ChatScript is a rule-based engine, where rules are created by humans writers in program scripts through a process called dialog flow scripting. These use a scripting metalanguage (simply called a "script") as their source code. Here what a ChatScript script file looks like:

```
#
# file: food.top
#
topic: ~food []

#! I like spinach
s: ( I like spinach ) Are you a fan of the Popeye cartoons?

    a: ( ~yes ) I used to watch him as a child. Did you lust after Olive Oyl?
        b: ( ~no ) Me neither. She was too skinny.
        b: ( yes ) You probably like skinny models.

    a: ( ~no ) What cartoons do you watch?
        b: ( none ) You lead a deprived life.
        b: ( Mickey Mouse ) The Disney icon.

#! I often eat chicken
u: ( ![ not never rarely ] I * ~ingest * ~meat ) You eat meat.

#! I really love chicken
u: ( !~negativeWords I * ~like * ~meat ) You like meat.

#! do you eat bacon?
?: ( do you eat _ [ ham eggs bacon] ) I eat '_0

#! do you like eggs or sushi?
```

```
?: ( do you like *_ or *_ ) I don't like '_0 so I guess that means I prefer '_1.

#! I adore kiwi.
s: ( ~like ~fruit ![~animal _bear] ) Vegan, you too...

#! do you eat steak?
?: ( do you eat _~meat ) No, I hate _0.

#! I eat fish.
s: ( I eat *_1 > )
    $food = '_0
    I eat oysters.
```

Above example mentioned in article [How to build your first chatbot using ChatScript](#).

## Basic Features

- Powerful pattern matching aimed at detecting meaning.
- Simple rule layout combined with C-style general scripting.
- Built-in WordNet dictionary for ontology and spell-checking.
- Extensive extensible ontology of nouns, verbs, adjectives, adverbs.
- Data as fact triples enables inferencing and supports JSON representation.
- Rules can examine and alter engine and script behavior.
- Remembers user interactions across conversations.
- Document mode allows you to scan documents for content.
- Ability to control local machines via popen/tcpopen/jsonopen.
- Ability to read structured JSON data from websites.
- Built in english pos-tagging and parsing
- Postgres and Mongo databases support for big data or large-user-volume chatbots.

## OS Features

- Runs on Windows or Linux or Mac or iOS or Android
- Fast server performance supports a thousand simultaneous users.
- Multiple bots can cohabit on the same server.

## Support Features

- Mature technology in use by various parties around the world.
- Integrated tools to support maintaining and testing large systems.
- UTF8 support allows scripts written in any language
- User support forum on [chatbots.org](http://chatbots.org)
- Issues or bugs on this repo

## Getting started

### Installation

Take this project and put it into some directory on your machine (typically we call the directory ChatScript, but you can name it whatever). That takes care of installation.

```
git clone https://github.com/ChatScript/ChatScript
```

### Standalone mode - run locally on a console (for development/test)

From your ChatScript home directory, go to the BINARIES directory:

```
cd BINARIES
```

And run the ChatScript engine

#### Windows

```
ChatScript
```

#### Linux

```
./LinuxChatScript64 local
```

Note: to set the file executable: `chmod a+x ./LinuxChatScript64`

#### MacOS

```
./MacChatScript local
```

This will cause ChatScript to load and ask you for a username. Enter whatever you want. You are then talking to the default demo bot **Harry**.

### Server Mode (for production)

From your ChatScript home directory, go to the BINARIES directory and run the ChatScript engine as server `###` Run the server on Windows

```
ChatScript port=1024
```

#### Run the server on Linux

```
./LinuxChatScript64
```

### Run the server on MacOS

```
./MacChatScript
```

This will cause ChatScript to load as a server.

But you also need a client (to test client-server communication). You can run a separate command window and go to the BINARIES directory and type

### Run a client (test) on Windows

```
ChatScript client=localhost:1024
```

### Run a client (test) on Linux

```
./LinuxChatScript64 client=localhost:1024
```

### Run a client (test) on MacOS

```
./MacChatScript client=localhost:1024
```

This will cause ChatScript to load as a client and you can talk to the server.

### How to build a bot

Run ChatScript locally. From the ChatScript command prompt, type

```
:build Harry
```

or whatever other preinstalled bot exists. If you have revised basic data, you can first:

```
:build 0
```

### How to compile the engine.

On windows if you have Visual Studio installed, launch VS2010/`chatscript.sln` or VS2015/`chatscript.sln` and do a build. The result will go in the BINARIES directory.

On Linux, go stand in the SRC directory and type `make server` (assuming you have make and g++ installed). This creates BINARIES/ChatScript, which can run as a server or locally. There are other make choices for installing PostGres or Mongo.

### Docker image

#### Building the base Docker image

The `Dockerfile` in this repository provides a ChatScript server with no bots. To build and run it, run the following commands:

```
docker build -t chatscript .
docker run -it -p 1024:1024 chatscript
```

Note: You will probably want to replace the image tag `chatscript` with a more meaningful one for your purposes.

## Building a Docker image containing bot data

Adding bot data to the base image above is as simple as writing a `Dockerfile` like the following one, which builds the `Harry` bot:

```
FROM chatscript

# Copy raw data needed for Harry
COPY ./RAWDATA/filesHarry.txt
COPY ./RAWDATA/HARRY /opt/ChatScript/RAWDATA/HARRY
COPY ./RAWDATA/QUIBBLE /opt/ChatScript/RAWDATA/QUIBBLE

# Build Harry
RUN /opt/ChatScript/BINARIES/LinuxChatScript64 local build1=filesHarry.txt
```

This `Dockerfile` can then be built and run in the same manner as the base `chatscript` image:

```
docker build -t chatscript-harry .
docker run -it chatscript-harry local
```

## Full Documentation

[ChatScript Wiki](#) (user guides, tutorials, papers)

## Contributing

1. Fork it
2. Create your feature branch (`git checkout -b my-new-feature`)
3. Commit your changes (`git commit -am 'Add some feature'`)
4. Push to the branch (`git push origin my-new-feature`)
5. Create new Pull Request

## Last releases

[changes.md](#)

## Author

- Bruce Wilcox

- home website: [BrilligUnderstanding.com](http://BrilligUnderstanding.com)
- mail: [gowilcox@gmail.com](mailto:gowilcox@gmail.com)