Pawn Documentation

Release 0.1.0

iLoveTux

September 10, 2016

Contents

Contents:

CHAPTER 1

Introduction

1.1 What is it and why do I care?

Pawn is a language built by mashing Python together with AWK and gluing them together with a little magic.

Simply put, Pawn is a language designed to scan a text file line-by-line looking for patterns and when a pattern matches, its associated action is executed. This is incredibly useful for parsing logs or other semi-structured, plain-text files.

I am a regular user of awk, but there are limitations to it. You are limited to the awk programming language which is a interpreted language inspired by C. With Pawn, you have complete access to Python and all the third-party modules currently installed.

A typical Pawn script will look like this:

```
BEGIN{
    import sys
    import requests

    critical_errors = 0
}
(?i)critical{
    requests.post("https://my-alerting-server", data=LINE)
    critical_errors += 1
}
END{
    print("There were {} critical errors found".format(critical_errors))
}
```

This script makes use of the special patterns BEGIN and END. BEGIN is executed once before processing any lines of the file. END is executed once after all the lines have been processed.

The pattern in the middle portion “(?i)critical” is a regex using Python’s inline-regex modifiers. This regex matches on any occurance of “critical” regardless of case.

The script essentialy looks for any line containing the word “critical” and when it finds one, it sends an http post request to an imaginary server (whose purpose is to respond to any critical events) and increment a counter. At the end of the script, a count of the lines containing the word “critical” is printed.

To execute this script, you would save it into a file called “critical_response.pawn” and run:

```
`pawn critical_response.pawn file.log`
```

1.2 Programatic API?

Pawn includes a programatic API as well for embeding its functionality into other applications. To use pawn in your application, you might do something like so:

```
from pawn import pawn

script = """
BEGIN{
    print("starting")
}
\d+{
    print("found a number")
}
END{
    print("ending")
}
"""

pawn(script=script, files=os.path.listdir("."))
```

This would run the script against each file in the current directory.

So, now for the magic:

Pawn accepts a script and a list of files. If a single file is passed in and it is not a list, it will be coerced into one. Once it is verified that we are working with a list, the list is scanned for strings, if a string is found in the list, it is assumed to be a filename and it will be opened. Once that is all done, we loop through the list of files and iterate through the files.

This is where the magic really happens since in Python file-objects are iterators which allow one to efficiently loop through the lines of a file. If we consider this, along with the above rules, we can pass any iterable yielding lines for processing.

1.3 How do I get it?

To get the latest version:

```
$ pip install https://github.com/ilovetux/pawn/archive/master.zip
```

For the nightlies:

```
$ pip install https://github.com/ilovetux/pawn/archive/dev.zip
```

1.4 How do I run the tests?

You can clone the repository and use the following command:

```
$ make test
```

or alternately:

```
$ python setup.py nosetests
```

1.5 What is this compatible with?

Pawn is tested and confirmed to work with

- Python 3.5
- Python 3.4
- Python 3.3
- Python 2.7
- pypy

Pawn should work on all platforms on which Python runs.

1.6 What is on the list to be done?

Check out our Issue Tracker for the items we are currently working on.

1.7 How can I help?

You can do all the github type things, submit an issue in our issue tracker or fork and submit a pull request. If none of that appeals to you, you can always send me an email personally at me@ilovetux.com

CHAPTER 2

API

`pawn.`**`pawn`**(*script*, *files*)

A simple interpreter for a language based on Python and AWK.

The rules are simple, and in a great act of blasphemy, they rely heavily on curly-braces "{}".

So, a pawn program consists of patterns and actions. Patterns are regular expressions which are evaluated and actions which consist of Python source code.

The main concept with pawn is that it is executed with two contexts the first is of a plain text input file and the second is a pawn script. Each line of the input is examined within the context of the pawn script. Based on the defined patterns a set of actions is built and executed in order.

The Python source code which comprises an action is evaluated with the following global variables defined:

- LINE: The current line being examined
- FIELDS: A list containing the fields into which the LINE was split
- FS: The field seperator, defaults to any whitespace

Parameters

- **`script`** (*`str`*) – The script to execute, can be passed as-is or a filename can be passed which will be read for the script
- **`files`** (*`str file list`*) – Can be a single file-like object or a str containing a filename or a list of either

Return type None

Returns None

CHAPTER 3

Introduction

3.1 What is it and why do I care?

Pawn is a language built by mashing Python together with AWK and gluing them together with a little magic.

Simply put, Pawn is a language designed to scan a text file line-by-line looking for patterns and when a pattern matches, its associated action is executed. This is incredibly useful for parsing logs or other semi-structured, plain-text files.

I am a regular user of awk, but there are limitations to it. You are limited to the awk programming language which is a interpreted language inspired by C. With Pawn, you have complete access to Python and all the third-party modules currently installed.

A typical Pawn script will look like this:

```
BEGIN{
    import sys
    import requests

    critical_errors = 0
}
(?i)critical{
    requests.post("https://my-alerting-server", data=LINE)
    critical_errors += 1
}
END{
    print("There were {} critical errors found".format(critical_errors))
}
```

This script makes use of the special patterns BEGIN and END. BEGIN is executed once before processing any lines of the file. END is executed once after all the lines have been processed.

The pattern in the middle portion “(?i)critical” is a regex using Python’s inline-regex modifiers. This regex matches on any occurance of “critical” regardless of case.

The script essentialy looks for any line containing the word “critical” and when it finds one, it sends an http post request to an imaginary server (whose purpose is to respond to any critical events) and increment a counter. At the end of the script, a count of the lines containing the word “critical” is printed.

To execute this script, you would save it into a file called “critical_response.pawn” and run:

```
`pawn critical_response.pawn file.log`
```

3.2 Programatic API?

Pawn includes a programatic API as well for embeding its functionality into other applications. To use pawn in your application, you might do something like so:

```
from pawn import pawn

script = """
BEGIN{
    print("starting")
}
\d+{
    print("found a number")
}
END{
    print("ending")
}
"""

pawn(script=script, files=os.path.listdir("."))
```

This would run the script against each file in the current directory.

So, now for the magic:

Pawn accepts a script and a list of files. If a single file is passed in and it is not a list, it will be coerced into one. Once it is verified that we are working with a list, the list is scanned for strings, if a string is found in the list, it is assumed to be a filename and it will be opened. Once that is all done, we loop through the list of files and iterate through the files.

This is where the magic really happens since in Python file-objects are iterators which allow one to efficiently loop through the lines of a file. If we consider this, along with the above rules, we can pass any iterable yielding lines for processing.

3.3 How do I get it?

To get the latest version:

```
$ pip install https://github.com/ilovetux/pawn/archive/master.zip
```

For the nightlies:

```
$ pip install https://github.com/ilovetux/pawn/archive/dev.zip
```

3.4 How do I run the tests?

You can clone the repository and use the following command:

```
$ make test
```

or alternately:

```
$ python setup.py nosetests
```

3.5 What is this compatible with?

Pawn is tested and confirmed to work with

- Python 3.5
- Python 3.4
- Python 3.3
- Python 2.7
- pypy

Pawn should work on all platforms on which Python runs.

3.6 What is on the list to be done?

Check out our Issue Tracker for the items we are currently working on.

3.7 How can I help?

You can do all the github type things, submit an issue in our issue tracker or fork and submit a pull request. If none of that appeals to you, you can always send me an email personally at me@ilovetux.com

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index