Task 1. Getting data from JIRA

In this part, you will extract the data you need to apply the data mining algorithms. The target is the Spring XD project (http://projects.spring.io/spring-xd/), which uses the JIRA project management tool (https://jira.atlassian.com/) to manage their issues.

This Task consist of two parts. In the first part (Task 1.1), you will create a dataset with the issues from the JIRA repository. In the second one (Task 1.2), you will pre-process and clean the dataset.

[INFO] Using the JIRA Python API

The JIRA Python API library (https://github.com/pycontribs/jira) eases the use of the JIRA REST API from Python and it has been used in production for years.


In [1]:

```python
""
Example of using JIRA Python API
""

from jira import JIRA

jira = JIRA('https://jira.atlassian.com')

issue = jira.issue('JRA-9')
print issue.fields.project.key  # 'JRA'
print issue.fields.issuetype.name  # 'Suggestion'
print issue.fields.reporter.displayName  # 'Mike Cannon-Brookes [Atlassian]'
```

JIRASERVER
Suggestion
Mike Cannon-Brookes

In [2]:

```
# Getting the keys of the last 3 reported issues

jira = JIRA('https://jira.spring.io')
print [issue.key for issue in jira.search_issues('project=XD order by created desc', maxRes=100)]

[u'XD-3768', u'XD-3767', u'XD-3766']
```

Task 1.1 Data extraction from JIRA

Issues

Using the JIRA Python API, connect to the Spring XD JIRA repository (https://jira.spring.io) and create a dataset with the following info about the issues:

- Key
- Assignee
- Creator
- Reporter
- Created
- Components
- Description
- Summary
- Fix Versions
- Subtask
- Issuetype
- Priority
- Resolution
- Resolution date
- Status
- Status Description
- Updated
- Versions
- Watches

- **Story Points** - **Hint**: the name of the field in which the story points are stored is not enough descriptive in this case. Thus, you might have a look at the data on each field and their distribution in order to understand where the story points are stored. Keep in mind that Story Points usually follow the fibonacci series (i.e. $\frac{1}{2}$, 1, 2, 3, 5, 8, 13, 20, 40, 100)

**Important**! Since the JIRA API allow you to connect directly with an online repository, there are some restrictions to avoid the overload of the servers. For example, JIRA API only allow for extracting bulks of issues (n=1000). Thus, it is important to take this into account when you are making request to some server.


In [3]:
```
# Connect with Spring
from jira import JIRA
jira = JIRA('https://jira.spring.io')
```

In [4]:
```
# it is a good idea to save the issues in a dataframe
import pandas as pd
issues = pd.DataFrame()
```
In [5]:

```python
# first solution
# the easy (and lazy) way

issues_in_proj_1 = jira.search_issues('project=XD', startAt=0, maxResults=1000)
issues_in_proj_2 = jira.search_issues('project=XD', startAt=1000, maxResults=1000)
issues_in_proj_3 = jira.search_issues('project=XD', startAt=2000, maxResults=1000)
issues_in_proj_4 = jira.search_issues('project=XD', startAt=3000, maxResults=1000)
```

In [6]:

```python
print len(issues_in_proj_1)
print len(issues_in_proj_2)
print len(issues_inProj_3)
print len(issues_inProj_4)
```

1000
1000
706

In [7]:

```python
# Second solution
# Getting all the issues in only one block

from jira import JIRA

jira = JIRA('https://jira.spring.io')

block_size = 1000
block_num = 0
allissues = []
while True:
    start_idx = block_num*block_size
    issues = jira.search_issues('project=XD', start_idx, block_size)
    if len(issues) == 0:
        # Retrieve issues until there are no more to come
        break
    block_num += 1
    for issue in issues:
        #log.info('%s: %s' % (issue.key, issue.fields.summary))
        allissues.append(issue)
```

In [8]:

```python
print 'Number of issues:', len(allissues)
```

Number of issues: 3706
In [9]:

```
## Into pandas

import pandas as pd

issues = pd.DataFrame()

for issue in allissues:
    d = {
        'key': issue.key,
        'assignee': issue.fields.assignee,
        'creator': issue.fields.creator,
        'reporter': issue.fields.reporter,
        'created': issue.fields.created,
        'components': issue.fields.components,
        'description': issue.fields.description,
        'summary': issue.fields.summary,
        'fixVersions': issue.fields.fixVersions,
        'subtask': issue.fields.issuetype.subtask,
        'issuetype': issue.fields.issuetype.name,
        'priority': issue.fields.priority.name,
        'resolution': issue.fields.resolution,
        'resolution.date': issue.fields.resolutiondate,
        'status.name': issue.fields.status.name,
        'status.description': issue.fields.status.description,
        'updated': issue.fields.updated,
        'versions': issue.fields.versions,
        'watches': issue.fields.watches.watchCount,
        'storypoints': issue.fields.customfield_10142
    }

    issues = issues.append(d, ignore_index=True)

print len(issues)

issues.head()
```

```
3706

Out[9]:
```

<table>
<thead>
<tr>
<th>assignee</th>
<th>components</th>
<th>created</th>
<th>creator</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>[]</td>
<td>2017-07-10T12:41:25.000+0000</td>
<td>abhineet kumar</td>
<td>The jobs that appear under Executions section ...</td>
</tr>
<tr>
<td>None</td>
<td>[]</td>
<td>2017-06-26T15:26:26.000+0000</td>
<td>Manuel Jordan</td>
<td>Working with Spring-XD version 1.3.2.RELEASE\n...</td>
</tr>
</tbody>
</table>
```
Save your results to a file

In [10]:

```python
issues.to_csv("issues-xd.csv", encoding='utf-8', header=True, index=False, line_terminator=\n```

### Extracting the changelog

There are two options to get the changelog.

1. Getting blocks of issues with additional information (e.g. the changelog)
2. Retrieve each issue individually with its additional information (e.g. the changelog)

If we opt for the first option (1)

```
issues = jira.search_issues('project=XD', start_idx, block_size, expand='changelog')
```

If we opt for the second option (2)

```
issue = jira.issue('FOO-100', expand='changelog')
```

Here, you have to extract data about the changes in the issues (number of fields to extract=6):

- Issue Key. The key of the issue that have been updated.
- Author: The key of the author who have made the change.
- Date. The timestamp indicating when the change have been done.
- Field. The updated field.
- From. The field value before the update.
- To. The field value after the update.
In [11]:

```python
import pandas as pd

changelog = pd.DataFrame()

for issue in allissues:
    issue = jira.issue(issue.key, expand='changelog')
    changelog = issue.changelog

    for history in changelog.histories:
        for item in history.items:
            d = {
                'key': issue.key,
                'author': history.author,
                'date': history.created,
                'field': item.field,
                'fieldtype': item.fieldtype,
                'from': getattr(item, 'from'), # because using item.from doesn't wor
                'fromString': item.fromString,
                'to': item.to,
                'toString': item.toString
            }

        changelog = changelog.append(d, ignore_index=True)

print "Number of records: ", len(changelog)
```

Number of records:  43222

In [12]:

```python
changelog.head()
```

Out[12]:

```
<table>
<thead>
<tr>
<th>author</th>
<th>date</th>
<th>field</th>
<th>fieldtype</th>
<th>from</th>
<th>fromString</th>
<th>key</th>
<th>to</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mark Pollack</td>
<td>2017-03-21T16:54:56.170+0000</td>
<td>Fix Version</td>
<td>jira</td>
<td>None</td>
<td>None</td>
<td>XD-3765</td>
<td>15491</td>
</tr>
<tr>
<td>Mark Pollack</td>
<td>2017-03-21T16:55:23.057+0000</td>
<td>status</td>
<td>jira</td>
<td>10000</td>
<td>To Do</td>
<td>XD-3765</td>
<td>10001</td>
</tr>
<tr>
<td>Mark Pollack</td>
<td>2017-03-21T16:55:23.057+0000</td>
<td>resolution</td>
<td>jira</td>
<td>None</td>
<td>None</td>
<td>XD-3765</td>
<td>8</td>
</tr>
<tr>
<td>Mark Pollack</td>
<td>2017-03-22T18:27:00.900+0000</td>
<td>summary</td>
<td>jira</td>
<td>None</td>
<td>Fix stream failover on YARN</td>
<td>XD-3765</td>
<td>None</td>
</tr>
<tr>
<td>Krzysztof Noga</td>
<td>2017-01-23T10:21:47.638+0000</td>
<td>Attachment</td>
<td>jira</td>
<td>None</td>
<td>None</td>
<td>XD-3763</td>
<td>23691</td>
</tr>
</tbody>
</table>
```

Save the dataset to a file
### Task 1.2 Pre-processing and cleaning the data

In this part, you will have to apply the following criteria to clean the dataset. The criteria consist of 10 cleaning steps.

- Remove all the issues with null or zero values
- Remove all the issues whose story points have been updated more than once, since they can represent misleading information. According to most of the estimation techniques, Story points must have been assigned once and never updated afterward.
In [17]:

# Filtering all the user stories that have been updated in the story points field
issues = df

to_remove = changelog[((changelog['field'] == 'Story Points') | (changelog['field'] == 'Actual Story Size') | (changelog['field'] == 'Effort points') | (changelog['field'] == 'Effort')) | (changelog['field'] == 'Points') & (changelog['fromString'].notnull())]

df = issues[~issues['key'].isin(to_remove['key'])]

print 'before/after =', len(issues), '/', len(df)

before/after = 3541 / 3064

- Remove all the issues that have not been addressed. We consider that an issue is addressed when its Status is set to Closed (or similar, e.g. Fixed, Completed) and its resolution field is set to Fixed (or similar, e.g. Done, Completed).

In [18]:

print issues['status.name'].unique()
print issues['resolution'].unique()

['To Do' 'Done' 'In PR' 'In Progress']
[nan 'Complete' 'Invalid' 'Works as Designed' 'Deferred' 'Won’t Fix' 'Cannot Reproduce' 'Incomplete' 'Duplicate' 'Fixed']

In [19]:

issues = df

df = issues[~issues['status.name'].isin(['Done']) | ~issues['resolution'].isin(['Complete'])]

print 'before/after =', len(issues), '/', len(df)

before/after = 3064 / 1035

- Remove all the issues whose Story Points or Description fields have been updated after they were addressed. Note that fields such as Title and Description may be adjusted or updated at any given time. However, once the issue is addressed updates rarely happen. Issue report that have been updated after they were addressed; they are likely to be unstable.

In [20]:

changelog.columns

Out[20]:
Index([u'author', u'date', u'field', u'fieldtype', u'from', u'fromString', u'key', u'to', u'toString'],
dtype='object')
In [21]:

```python
issues = df
to_remove = []
for ix, line in issues.iterrows():
    date = pd.to_datetime(line['resolution.date'])
    key = line['key']
    if (pd.notnull(date)):
        key_remove = changelog[(changelog['key'] == key) &
                                ((changelog['field'] == 'description') |
                                 (changelog['field']
                                  .to_datetime(changelog['date']) > date))][
                                 'key']
        to_remove.append(key_remove)

df = issues[ ~issues['key'].isin(to_remove) ]
print('before/after =', len(issues), '/', len(df))
```

```
C:\Anaconda2\envs\gl-env\lib\site-packages\pandas\core\dtypes\missing.py:28
9: RuntimeWarning: tp_compare didn't return -1 or -2 for exception
  if left.shape != right.shape:
before/after = 1035 / 1035

- Remove all the issues whose story points are not in the Fibonacci series (i.e. ½, 1, 2, 3, 5, 8, 13, 20, 40, 100)
```

In [22]:

```python
issues = df
FIBO_CARDS=[0.5, 1, 2, 3, 5, 8, 13, 20, 40, 100]
df = issues[ issues['storypoints'].isin(FIBO_CARDS) ]
print('before/after =', len(issues), '/', len(df))
```

```
before/after = 1035 / 943

- Remove all the issues whose informative fields are updated after the story points initialization, since they are considered as unstable issues. We define informative fields: Issue Type, Description, Summary, and Component/s.
```

In [23]:

```python
changelog['field'].unique()
```

```
Out[23]:
array(['Fix Version', 'status', 'resolution', 'summary', 'Attachment',
       'description', 'Version', 'Component', 'priority', 'issuetype',
       'Rank', 'labels', 'Link', 'Pull Request URL', 'assignee', 'Sprint',
       'Epic Link', 'Actual Story Points', 'Acceptance Criteria',
       'Story Points', 'RemoteIssueLink', 'Parent', 'Comment',
       'Epic Child', 'Epic Name', 'Parent Issue', 'Epic Colour',
       'reporter', 'Reference URL', 'environment', 'Workflow', 'Key',
       'project', 'Out of Scope', 'Epic Status', 'Time Spent', 'WorklogId',
       'timespent', 'timeestimate', 'Project'], dtype=object)
```
In [24]:

```python
issues = df

to_remove = []
for ix, line in issues.iterrows():
    date = pd.to_datetime(line['created'])
    key = line['key']
    key_remove = changelog[(changelog['key'] == key) &
                            (changelog['field'] == 'description') |
                            (changelog['field'] == 'summary') |
                            (changelog['field'] == 'Cc') &
                            (pd.to_datetime(changelog['date']) > date)]['key']
    to_remove.append(key_remove)

df = issues[~issues['key'].isin(to_remove)]

print 'before/after =', len(issues), '/', len(df)
```

```
before/after = 943 / 943
```

- Remove outliers. Use Tukey's fences.

In [25]:

```python
issues = df

import numpy as np

def remove_outliers(df, column, k = 1.5):
    first_quartile = np.percentile(df[column], 25)
    third_quartile = np.percentile(df[column], 75)
    iqr = third_quartile - first_quartile

    minval = first_quartile - k*iqr
    maxval = third_quartile + k*iqr

    return df[(df[column] > minval) | (df[column] < maxval)].copy()

# issues = issues[~pd.isnull(issues['storypoints'])]

df = remove_outliers(df, 'storypoints')

print 'before/after =', len(issues), '/', len(df)
```

```
before/after = 943 / 943
```

- Remove the stop words from title and the summary.
Remove the code snippets from the title and the summary of the issues. The code snippets can be easily identify by looking for the tag `<code></code>`.

<table>
<thead>
<tr>
<th></th>
<th>description</th>
<th>summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The jobs appear Executions section Jobs spring...</td>
<td>How I make job restartable spring xd</td>
</tr>
<tr>
<td>1</td>
<td>Working Spring-XD version 1.3.2.RELEASE Starti...</td>
<td>admin config timezone command work</td>
</tr>
</tbody>
</table>

- Remove the code snippets from the title and the summary of the issues. The code snippets can be easily identify by looking for the tag `<code></code>`.
In [29]:

```python
pd.options.display.max_colwidth = -1

# the issue report XD-3751 has code in its description
print original[ original['key'] == 'XD-3751']['description']

print
# print after
print issues[ issues['key'] == 'XD-3751']['description']
```

17 In a case where reactor's ringbuffer is full and thus handling backpressure blocking `onNext` phase where `onComplete` is send will cause a deadlock. This is shown by a thread dump during a shutdown. This will basically break further deployments for this stream in distributed mode while single node will show more errors during undeployment. This is shown by a thread dump during a shutdown. This will basically break further deployments for this stream in distributed mode while single node will show more errors during undeployment.

```java
[pool-7-thread-1] #58 prio=5 os_prio=0 tid=0x979fe800 nid=0x54de runnable [0x986ad000]
   java.lang.Thread.State: TIMED_WAITING (parking)
     at sun.misc.Unsafe.park(Native Method)
     at java.util.concurrent.locks.LockSupport.parkNanos(LockSupport.java:338)
     at reactor.jarjar.com.lmax.disruptor.RingBuffer.next(RingBuffer.java:246)
     at reactor.core.processor.RingBufferSubscriberUtils.onNext(RingBufferSubscriberUtils.java:30)
     at reactor.core.processor.RingBufferProcessor.onNext(RingBufferProcessor.java:575)

[main-EventThread] #19 daemon prio=5 os_prio=0 tid=0x9b93a400 nid=0x54b1 runnable [0x9aefe000]
   java.lang.Thread.State: TIMED_WAITING (parking)
     at sun.misc.Unsafe.park(Native Method)
     at java.util.concurrent.locks.LockSupport.parkNanos(LockSupport.java:338)
     at reactor.jarjar.com.lmax.disruptor.RingBuffer.next(RingBuffer.java:246)
     at reactor.core.processor.RingBufferSubscriberUtils.onNext(RingBufferSubscriberUtils.java:54)
     at reactor.core.processor.RingBufferProcessor.onNext(RingBufferProcessor.java:54)
     at org.springframework.xd.greenplum.gpfdist.GPFDistMessageHandler.doStop(GPFDistMessageHandler.java:170)
```

I've been crafting workaround for this by trying to wait reactor stream/buffer to get drained by gpdb and finally as last resort, forcing processor in reactor to shutdown.

Name: description, dtype: object

17 In case reactor's ringbuffer full thus handling backpressure blocking `onNext`, shutdown phase `onComplete` send cause deadlock. This shown thread dump shutdown. This basically break deployments stream distributed mode single node show errors undeployment. I've been crafting workaround trying wait reactor stream/buffer to get drained by gpdb and finally last resort, forcing processor in reactor to shutdown.

Name: description, dtype: object

- Remove all the completed issues whose don't have any assignee.
In [30]:

def = issues[issues['assignee'].notnull()]

print 'before/after =', len(issues), '/', len(df)

before/after = 869 / 222

Finally, save the cleaned dataset to a file.

In [31]:

issues = df

issues.to_csv("issues-xd-cleaned.csv", sep=',', encoding='utf-8', doublequote = True, header=True)

Task 2 Getting data from Github

In this task, you will extract data from the same project used in Task 1 (the Spring-XD project), but now the source will be the GitHub repository instead of the JIRA project management tool.

Like in Task 1, this Task consist of two parts. In the first part (Task 2.1), you will create a dataset from the repository whereas in the second one (Task 2.2) you will pre-process and clean the dataset.

[INFO] Creating a GitHub API Connection

Since GitHub implements OAuth, getting API access involves creating an account followed by one of two possibilities:

1. creating an application to use as the consumer of the API or
2. creating a Personal Access Token that will be linked directly to your account

You can create your 'Personal API Access Token' from the Settings menu in your GitHub account.
https://github.com/settings/tokens

We’ll opt to take advantage of a Python library so that we can avoid some of the tedious details involved in making requests, parsing responses, and handling pagination. In this particular case, we’ll use PyGithub, which can be installed with the somewhat predictable pip install PyGithub.

https://github.com/PyGithub/PyGithub
In [32]:

```python
from github import Github

# XXX: Specify your own access token here
ACCESS_TOKEN = '08e610638148003f5098f191324375149ee14d72'

# Specify a username and repository of interest for that user.
USER = 'spring-projects'
REPO = 'spring-xd'

client = Github(ACCESS_TOKEN, per_page=100)
user = client.get_user(USER)
repo = user.get_repo(REPO)

print "Rate limit remaining", client.rate_limiting
# print "Rate Limit reached. Reset time: ", datetime.fromtimestamp(client.rate_limiting_rese
```

**Task 2.1 Extracting the data**

Using the [PyGithub](https://github.com/PyGithub/PyGithub) API, connect to the Spring XD Github repository and extract the following data:

- Contributors: a list of users who contribute to the project
- Repositories: a list of repositories in which the contributors are involved
- Contributors' stats: Weekly stats from the contributors
  - Repository
  - Week
  - Number of additions
  - Number of deletions
  - Number of commits
  - Contributor
- Commits:
  - Author
  - Date
  - Changes
  - Additions
  - Deletions
  - Filename
  - Comment_count
  - Message

**Contributors**
In [33]:

    # Get a list of people who have contributed to the repo.
    contributors = [ s for s in repo.get_contributors() ]

    print "Number of contributors: ", len(contributors)

Number of contributors: 44

Contributors' stats

Stats from contributors
http://pygithub.readthedocs.io/en/latest/github_objects/StatsContributor.html#statscontributor
(http://pygithub.readthedocs.io/en/latest/github_objects/StatsContributor.html#statscontributor)

In [36]:

    stats_contributors = repo.get_stats_contributors() 

    stats_contributors_df = pd.DataFrame() 
    for sc in stats_contributors: 
        for w in sc.weeks: 
            d = { 
                "week" : w.w, 
                "additions" : w.a, 
                "deletions" : w.d, 
                "commits" : w.c, 
                "author" : sc.author.login, 
                "total" : sc.total 
            } 
            stats_contributors_df = stats_contributors_df.append(d, ignore_index=True)

In [37]:

    print "Size: ", len(stats_contributors_df)
    stats_contributors_df.head()

Size: 11616

Out[37]:

<table>
<thead>
<tr>
<th>additions</th>
<th>author</th>
<th>commits</th>
<th>deletions</th>
<th>total</th>
<th>week</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>LinkedList</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
<td>2013-04-07</td>
</tr>
<tr>
<td>0.0</td>
<td>LinkedList</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
<td>2013-04-14</td>
</tr>
<tr>
<td>0.0</td>
<td>LinkedList</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
<td>2013-04-21</td>
</tr>
<tr>
<td>0.0</td>
<td>LinkedList</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
<td>2013-04-28</td>
</tr>
<tr>
<td>0.0</td>
<td>LinkedList</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
<td>2013-05-05</td>
</tr>
</tbody>
</table>

Save to a file
In [38]:
# Saving to a file
stats_contributors_df.to_csv("stats-contributors-xd.csv", index=False, encoding='utf-8')

In [39]:
# Aggregated stats
stats_contributors_df[['additions', 'author', 'commits', 'deletions']].groupby(by='author')

Out[39]:

<table>
<thead>
<tr>
<th></th>
<th>additions</th>
<th>commits</th>
<th>deletions</th>
</tr>
</thead>
<tbody>
<tr>
<td>author</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BoykoAlex</td>
<td>383.0</td>
<td>2.0</td>
<td>84.0</td>
</tr>
<tr>
<td>LinkedList</td>
<td>5.0</td>
<td>1.0</td>
<td>5.0</td>
</tr>
<tr>
<td>aclement</td>
<td>13151.0</td>
<td>19.0</td>
<td>2109.0</td>
</tr>
<tr>
<td>aeisenberg</td>
<td>49221.0</td>
<td>4.0</td>
<td>98.0</td>
</tr>
<tr>
<td>agandhinit</td>
<td>244.0</td>
<td>1.0</td>
<td>69.0</td>
</tr>
</tbody>
</table>

Repositories
In [40]:

repositories = pd.DataFrame()
for i, c in enumerate(contributors):
    print(f"Contributor: {c.login}

    repos = c.get_repos()

    print(repos)

    for r in repos:
        d = {
            'contributor': c.login,
            'lang': r.language,
            'owner': r.owner.login,
            'repo': r.name
        }
        repositories = repositories.append(d, ignore_index=True)
    print(f"Processed {i+1} contributors.
print(f"Rate limit remaining: {client.rate_limiting}

print(f"Repositories: {len(repositories)}")
repositories.head()

Rate limit remaining: (4957, 5000)
Contributor: artembilan
<github.PaginatedList.PaginatedList instance at 0x0000000046181C88>
Processed 27 contributors.
Rate limit remaining: (4955, 5000)
Contributor: BoykoAlex
<github.PaginatedList.PaginatedList instance at 0x00000000478C8A88>
Processed 28 contributors.
Rate limit remaining: (4954, 5000)
Contributor: gregturn
<github.PaginatedList.PaginatedList instance at 0x0000000046831048>
Processed 29 contributors.
Rate limit remaining: (4952, 5000)
Contributor: kdowbecki
<github.PaginatedList.PaginatedList instance at 0x0000000046E77448>
Processed 30 contributors.
Rate limit remaining: (4951, 5000)
Contributor: morfeo8marc
<github.PaginatedList.PaginatedList instance at 0x0000000046267408>
Processed 31 contributors.

Save to a file

In [41]:

# save to file
repositories.to_csv("contributors-xd.csv", index=False, encoding='utf-8')

Commits

In [42]:

commitsrepo = repo.get_commits()
In [43]:

import pandas as pd

commits = pd.DataFrame()

for i, commit in enumerate(commitsrepo):
    try:
        d = {
            'author': commit.author.login if commit.author.login is not None else commit.author.name,
            'changes': commit.stats.total,
            'additions': commit.stats.additions,
            'deletions': commit.stats.deletions,
            'files': [f.filename for f in commit.files],
            'author_date': commit.commit.author.date,
            'commiter_date': commit.commit.committer.date,
            'commiter': commit.commit.committer.name,
            'message': commit.commit.message,
            'comments': [comment for comment in commit.get_comments()]
        }
        commits = commits.append(d, ignore_index=True)
    except Exception, e: #ssl.SSLError:
        print "Encountered an error fetching details of commit ", i, ", Skipping."
        print e

print "Number of commits: ", len(commits)

commits.head()

Encountered an error fetching details of commit # 1361  Skipping.
'NoneType' object has no attribute 'login'
Encountered an error fetching details of commit # 1362  Skipping.
'NoneType' object has no attribute 'login'
Encountered an error fetching details of commit # 1363  Skipping.
'NoneType' object has no attribute 'login'
Encountered an error fetching details of commit # 1364  Skipping.
'NoneType' object has no attribute 'login'
Encountered an error fetching details of commit # 1365  Skipping.
'NoneType' object has no attribute 'login'
Encountered an error fetching details of commit # 1369  Skipping.
'NoneType' object has no attribute 'login'
Encountered an error fetching details of commit # 1382  Skipping.
'NoneType' object has no attribute 'login'
Encountered an error fetching details of commit # 1385  Skipping.
'NoneType' object has no attribute 'login'
Encountered an error fetching details of commit # 1386  Skipping.
'NoneType' object has no attribute 'login'

Save to a file

In [44]:

# save to file
commit.to_csv("commits-xd.csv", index=False, encoding='utf-8')

---

Task 2.2 Pre-processing and cleaning the data

Apply the following criteria to clean the dataset:
- Remove commits with null message
- Remove all the commits related to artifact release -- hint: the message begins with [artifact-release]
- Remove commits from authors who have been commit only once
- Remove all the commits from changes that have been done to auxiliar files (i.e. gradle.properties, README, docfiles)
- Remove all the commits that are related to a given release

In [80]:

        commits = pd.read_csv("commits-xd.csv")

In [81]:

        commits.head()

Out[81]:

<table>
<thead>
<tr>
<th></th>
<th>additions</th>
<th>author</th>
<th>author_date</th>
<th>changes</th>
<th>comments</th>
<th>commiter</th>
<th>commiter_date</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.0</td>
<td>spring-buildmaster</td>
<td>2017-03-22 19:23:27</td>
<td>2.0</td>
<td>[]</td>
<td>Spring Buildmaster</td>
<td>2017-03-22 19:23:27</td>
</tr>
<tr>
<td>1</td>
<td>1.0</td>
<td>spring-buildmaster</td>
<td>2017-03-22 19:23:23</td>
<td>2.0</td>
<td>[]</td>
<td>Spring Buildmaster</td>
<td>2017-03-22 19:23:23</td>
</tr>
<tr>
<td>2</td>
<td>5.0</td>
<td>jvalkeal</td>
<td>2017-02-01 14:21:38</td>
<td>10.0</td>
<td>[]</td>
<td>Janne Valkealahti</td>
<td>2017-02-01 14:21:38</td>
</tr>
<tr>
<td>3</td>
<td>1.0</td>
<td>markpollack</td>
<td>2016-12-23 15:06:47</td>
<td>2.0</td>
<td>[]</td>
<td>Mark Pollack</td>
<td>2016-12-23 15:06:47</td>
</tr>
<tr>
<td>4</td>
<td>4.0</td>
<td>markpollack</td>
<td>2016-12-23 15:06:15</td>
<td>8.0</td>
<td>[]</td>
<td>Mark Pollack</td>
<td>2016-12-23 15:06:15</td>
</tr>
</tbody>
</table>

1 - Remove commits with null message

In [82]:

        print 'before', len(commits)
        commits = commits[ ~pd.isnull(commits['message']) ]
        print 'after', len(commits)

before 2097
after 2093

2 - Remove all the commits related to artifact release -- hint: the message begins with [artifact-release]
3 - Remove commits from authors who have been commit only once

```python
In [83]:
print 'before', len(commits)
commits = commits[commits['message'].str.startswith('[artifact-relea
print 'after', len(commits)
```
In [90]:
stats['author'].unique()

Out[90]:
array(['LinkedList', 'sworisbreathing', 'htynkn', 'philwebb', 'agandhinit',
      'sathiyas', 'parikhkc', 'chrisjs', 'kashyap-parikh', 'artembilan',
      'sabbyanandan', 'BoykoAlex', 'morfeo8marc', 'kdowbecki',
      'thomasdarimont', 'nebhale', 'wilkinsonsona', 'fbiville', 'datianshi',
      'gregturn', 'fmarchand', 'smaldini', 'twoseat', 'aeisenberg',
      'luijiong1982', 'jbrisbin', 'pperalta', 'aclement', 'dsyer',
      'jvalkeal', 'jencompgeek', 'mmnella', 'spring-buildmaster',
      'mbogoevici', 'cppwfs', 'dturanski', 'tekul', 'ghillert',
      'erichottard', 'markpollack', 'garyrussell', 'trisberg',
      'markfisher', 'ilayaperumalg'], dtype=object)

In [91]:
authors_commits = stats.groupby(by='author')['commits'].sum()

In [92]:
authors_commits = authors_commits.reset_index()

In [93]:
authors_commits.head()

Out[93]:
<table>
<thead>
<tr>
<th>author</th>
<th>commits</th>
</tr>
</thead>
<tbody>
<tr>
<td>BoykoAlex</td>
<td>2.0</td>
</tr>
<tr>
<td>LinkedList</td>
<td>1.0</td>
</tr>
<tr>
<td>aclement</td>
<td>19.0</td>
</tr>
<tr>
<td>aeisenberg</td>
<td>4.0</td>
</tr>
<tr>
<td>agandhinit</td>
<td>1.0</td>
</tr>
</tbody>
</table>

In [94]:
authors_commits[authors_commits['commits'] == 1][['author']]

Out[94]:
<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>17</td>
</tr>
<tr>
<td>22</td>
</tr>
<tr>
<td>31</td>
</tr>
<tr>
<td>32</td>
</tr>
<tr>
<td>35</td>
</tr>
<tr>
<td>38</td>
</tr>
</tbody>
</table>
Name: author, dtype: object
In [95]:

print 'before', len(commits)

cs = authors_commits[authors_commits['commits'] == 1]['author']

commits = commits[~commits['author'].isin(cs) ]

print 'after', len(commits)

before 2064
Number of authors to delete from the stats:  9
after 2055

In [96]:

commits['author'].unique()

Out[96]:
array(['jvalkeal', 'markpollack', 'garyrussell', 'cppwfs', 'mbogoevici',
      'ilayaperumal', 'artembilan', 'dturanski', 'sabbyanandan',
      'ghillert', 'trisberg', 'mminella', 'pperialta', 'aclement',
      'BoykoAlex', 'ericbottard', 'morfeo8marc', 'fmarxand', 'smaldini',
      'kdowbecki', 'twoseat', 'jbrisbin', 'thomasdarimont', 'markfisher',
      'nebhale', 'spring-buildmaster', 'liujiong1982', 'wilkinsona',
      'dsyer', 'fbiville', 'tekul', 'aeisenberg', 'jencompgeek',
      'datianshi', 'gregturn'], dtype=object)

4 - Remove all the commits from changes that have been done to auxiliar files (i.e. gradle.properties, README, docfiles)

In [97]:

to_remove = []

for ix, line in commits.iterrows():
    if ('gradle.properties' in line['files']) | ('README' in line['files']):
        commits = commits.drop(index=ix)

print 'after', len(commits)

after 1967
In [98]:
commits.head()

Out[98]:

<table>
<thead>
<tr>
<th>additions</th>
<th>author</th>
<th>author_date</th>
<th>changes</th>
<th>comments</th>
<th>commiter</th>
<th>commiter_date</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>jvalkeal</td>
<td>2017-02-01 14:21:38</td>
<td>10.0</td>
<td>[]</td>
<td>Janne Valkealahti</td>
<td>2017-02-01 14:21:38</td>
</tr>
<tr>
<td>4</td>
<td>markpollack</td>
<td>2016-12-23 15:06:15</td>
<td>8.0</td>
<td>[]</td>
<td>Mark Pollack</td>
<td>2016-12-23 15:06:15</td>
</tr>
<tr>
<td>5</td>
<td>garyrussell</td>
<td>2016-05-03 18:13:30</td>
<td>36.0</td>
<td>[]</td>
<td>Artem Bilan</td>
<td>2016-11-01 16:09:41</td>
</tr>
<tr>
<td>6</td>
<td>garyrussell</td>
<td>2016-02-29 20:26:42</td>
<td>134.0</td>
<td>[]</td>
<td>Artem Bilan</td>
<td>2016-03-01 00:39:36</td>
</tr>
</tbody>
</table>

5 - Remove all the commits that are related to a given release

In [99]:

commits = commits[~commits['message'].str.startswith('[Release'] )]

print 'after', len(commits)

after 1967
Finally, save the cleaned dataset to a file.

In [100]:

    # save to file
    commits.to_csv("commits-xd-cleaned.csv", index=False, encoding='utf-8')