UNIVERSIDADE FEDERAL DO CARIRI
FACULDADE DE CIÊNCIA DA COMPUTAÇÃO
DEPARTAMENTO DE TECNOLOGIA

Disciplina: CC0016 e MC0012 Laboratório de Programação

Aluno:
Wanderley de Macedo Beserra Filho

Professor responsável:
Paola Rodrigues de Godoy Accioly

Mineração de Repositórios GitHub

Da coleta:

1 – Objetivos: O trabalho principal tem de interesse usar a API (*Application Programming Interface*) do GitHub, e fazer a coleta, dentre um filtro indicado nos repositórios, das 5 principais linguagens Java, C, Python, Ruby, Javascript, e os 200 autores com as maiores contribuições por *Pull Requests* daquele repositório.

2 – Procedimentos Experimentais: De início, para o desenvolvimento do *script*, foi usado o *shell* do *linux*, em um ambiente simulado para emulação de computador em mobile. Porém, a exclusão deste, faz-se possível repetir o mesmo resultado. De forma arbitraria, foi escolhido um ponto de partida, e em seguida, obedecendo a filtragem dos repositórios com x > 1000 estrelas que contenha ao menos 200 *Pull Requests* como *Closed*, foram selecionados 4 repositórios de cada uma das seguintes linguagens:

1. *Java;*
2. *C;*
3. *Python;*
4. *Ruby;*
5. *Javascript.*

As requisições aos repositórios, e toda a documentação teórica e semi prática é possível ser encontrada em https://docs.github.com/pt/rest. Na aba de *Pull Requests*, tem-se uma relação de links e parâmetros que serão usados com importância no projeto. São alguns deles *State*, os quais podem ser *open* , *closed* ou *all*, será escolhido o *closed* para filtrar apenas os *Pull Requests* em estado “fechado”. A ordem esta, por padrão, dos mais recentes (*created*). Mais a frente será necessário ainda, para obter os 200 colaboradores exatos, mais dois parâmetros o *per_page e page* . É necessário lembrar, que para futura analise de dados obtidos, será usado a extensão *.csv*, que são basicamente dados separados por virgulas (e podem ser facilmente analisados no *Excel* por exemplo) .

4 – Resultados e Discussão:

Pode-se notar, nos repositórios, que foi pensado a manipulação automática de um arquivo os quais erão feitas as requisições pelo comando *curl* do *Shell* e seus consequentes parâmetros. Após criado o repositorios.txt, no qual estão ordenados e prontos como : os donos, repositórios e respectivas linguagens:

```
Ex.: Linguagem   dono   nome_do_repositório
```

1. Das amostras obtidas:

```
JAVA iluwatar java-design-patterns
JAVA elastic elasticsearch
JAVA spring-projects spring-boot
JAVA ReactiveX RxJava
C torvalds linux
C netdata netdata
C redis redis
C git git
PYTHON donnemartin system-design-primer
PYTHON public-apis public-apis
PYTHON TheAlgorithms Python
PYTHON vinta awesome-python
RUBY rails rails
RUBY jekyll jekyll
RUBY discourse discourse
RUBY fastlane fastlane
JAVASCRIPT freeCodeCamp freeCodeCamp
JAVASCRIPT vuejs vue
JAVASCRIPT facebook react
JAVASCRIPT twbs bootstrap
```

Com todas as amostras necessárias no arquivo de texto, agora basta a criação de um *script* autossuficiente que manipule cada linha , faça as requisições, filtragem e gere os *.csv* como nomes próprios, toma-se o exemplo do primeiro item da lista:

```
java_iluwatar_java-design-patterns.csv
```

Ao final, serão coletados 20 arquivos iguais a este, 4 de cada linguagem (*5).

2. Da criação do script:

No ambiente *linux* é possível desenvolver *scripts* que rodem da mesma forma no terminal, porém de forma conjunta de comandos, para isso, basta editar o código, e salvá-lo como *.sh* . Usaremos aqui o *VIM*, editor via terminal do linux:

```
:~$ vi script.sh
```

Em seguida, vamos dar permissão ao *script* que seja manipulável e executável posteriormente , entrando no modo de comando apertando "*Esc*" e logo em seguida ":", sem as aspas. Basta digitar *"!chmod777%"* e pronto. Entraremos agora no modo inserção "i", e para definir agora, qual interpretador de comandos sera usado para executar o *script* que no caso será o shell.

```
#!/bin/shell
```

Logo após, devido a uma limitação da taxa de requisições, é necessário gerar um *token* para obter mais de 5000/hora. Isto deve ser feito no perfil do usuário pelo *GitHub* na pagina https://github.com/settings/tokens. Se o script for executado de primeira, não será necessário o uso dele, usaremos para estudo e segurança.

Criaremos uma variável agora que receberá o *token* gerado, que deve ser armazenado em um arquivo *.txt* para não ser visível na linha principal do script assim:

```
token=$ (head     -n 1 "/home/diretorio/TokenDoGItHub")
```

O comando *head -n 1*, irá pegar apenas a primeira linha do diretório entre aspas, e por meio do *$()*, a saída será armazenada na variável "*token*".

Logo em seguida vamos manipular cada linha do arquivo com os nomes e repositórios, será feita a coleta por meio de um *Loop* com o comando *read*, que irá ler criando 3 variáveis para cada espaço em branco na linha, além de uma quarta variável que será criada juntando as três com "_" e será utilizada mais para frente.

```
while read linguagem dono repositorio; do
      lin_don_rep="${linguagem}_${dono}_${repositorio}"
            > requesicao2.txt
done < "/home/diretorio/repositorios"
```

Com isso temos nosso *script* principal pronto. Agora vamos fazer as requisições dos repositórios, usando os parâmetros já comentados, e armazenar em formato *JSON* (uma linguagem de formato compacto) em um arquivo de texto "requesicao2.txt". Além disso, será necessária mais de uma requisição, pois temos o parâmetro *per_page* limitado com no máximo 100 P*ull Requests*, assim, criaremos um "*for*" onde a variável "i", ira acrescentar duas vezes para cada pagina. O que significa que teremos duas paginas com um total de 200 *Pull Requests*.

```
for i in 1 2; do

curl \
 -u "derleymad":$token" \
 -H "Accept: application/-vnd.github.v3+json" \
 "https://api.github.com/repos/$dono/$repositorio/pulls?
state=closed&page=$i&per_page=100" >> requesicao2.txt
```

```
done
```

Temos agora um *script* funcional que requisita e armazena informações em *JSON*, basta agora filtrar, usando os comandos "*grep*", "*sed*", "*cut*" e "*paste*". Para identificar qual o nome do usuário que fez o *Pull Request*, nos meus estudos, foi identificado algumas estruturas que sempre aparecem de forma padrão:

Ex.1:

```
"number":
"state":
"locked":
"title":
"user": {
  "login": "nome_a_ser_filtrado",
```

Assim, o que queremos é dentro da estrutura "*user*", retirar apenas a linha adjacente com a especie "*login*": "***nome_a_ser_filtrado***". Para isso após alguma pesquisa por metacaracteres e parâmetros para *grep* e *sed* decidi:

Usando `grep -A NUM` temos *num* como numero a ser passado de linhas que serão filtradas após a palavra-chave. Pois fazendo desta forma eliminamos todos os formatos e *users, title* e *login* existentes que não são relevantes para busca:

```
grep -A 6 number requesicao.txt
```

Isso, irá me retornar todas as 6 linhas abaixo do filtro "*number*". Agora faremos isso até o login um servindo de entrada para o outro usando o operador | (*pipe)*, além de ir diminuindo o *range* do parâmetro -A.

```
grep -A 6 number requesicao.txt | grep -A 5 state | grep -A 4
locked | grep -A 3 title | grep login
```

Temos então até aqui a seguinte filtragem com resposta:

```
"login": "nome_a_ser_filtrado",
```

Tem-se agora o tratamento com *sed* para retirar os caracteres indesejados:

```
sed 's/"login": "//g' | sed 's/",//g'
```

Por conseguinte, foi decidido dividir a filtragem em dois arquivos, um com apenas os nomes, e outro com apenas os números, para depois concatenar ambos os dados no formato desejado. Para ter esse resultado, com o *username* correto, podemos ter mais de uma incidência de um mesmo nome, de forma não consecutiva, portanto, temos de usar o comando *sort* para ordenar os nomes por ordem alfabeta, e assim, permitir que o comando *uniq -c*, remova as duplicações corretamente fazendo a contagem com o parâmetro *-c*.

Vale lembrar que o *sort -n -r* vem para reverter a ordem para decrescente de ocorrências.

```
sort | uniq -c | sort -n -r
```

Por fim, para selecionar os nomes, o comando *cut -c "x – y"*, irá fazer o corte do intervalo de colunas definido em *x* e *y,* neste último caso um limite máximo ao nome*:*

```
cut -c 15-100 > apenasnumeros.txt
```

Agora repetimos o processo para filtragem dos números apenas:

```
grep -A 6 number requesicao2.txt | grep -A 5 state | grep -A4 locked |
grep -A 3 title | grep login | sed 's/"login": "//g' |sed 's/",//g' |sort
| uniq -c | sort -n -r | cut -c 5-10 > apenasnumeros.txt
```

Com os dois arquivos já criados, é necessária concatenar agora ambos com o comando *paste* , o parâmetro *-d,* indica qual separador de colunas será usado, como o arquivo final estará em *.csv* , será , pois, a virgula, além do *rm -f *.txt* para apagar todos os *.txt* temporarios:

```
paste -d "," apenasnomes.txt apenasnumeros.txt
"/home/diretorio/$lin_rep.csv"

rm -f *.txt
```

5 – Conclusões: Com todos os comandos prontos, o *script* será montado, e portanto, temos novamente de voltar ao modo de comando do vim, e digitaremos *:wq* , (*write* and *quit*), que irá salvar o script, e sair do vim.

Agora basta executar o arquivo no *shell* com o comando:

```
:~$ ./script.sh
```

Script Pronto:

```
#!/bin/shell

token=$ (head     -n 1 "/home/diretorio/TokenDoGItHub")

while read linguagem dono repositorio; do
```

```
lin_rep="${linguagem}_${repositorio}"

        > requesicao2.txt

        for i in 1 2; do

        curl \
        -u "derleymad":$token" \
        -H "Accept: application/-vnd.github.v3+json" \
        "https://api.github.com/repos/$dono/$repositorio/pulls?
state=closed&page=$i&per_page=100" >> requesicao2.txt

        done

        grep -A 6 number requesicao.txt | grep -A 5 state | grep -A 4
locked | grep -A 3 title | grep login | sed 's/"login": "//g' | sed
's/",//g' | sort | uniq -c | sort -n -r | cut -c 15-100 >
apenasnumeros.txt
        grep -A 6 number requesicao2.txt | grep -A 5 state | grep -A4
locked | grep -A 3 title | grep login | sed 's/"login": "//g' |sed
's/",//g' |sort | uniq -c | sort -n -r | cut -c 5-10 > apenasnumeros.txt
        paste -d "," apenasnomes.txt apenasnumeros.txt
"/home/diretorio/$lin_rep.csv"

done < "/home/diretorio/repositorios"
```

7 – Analise das Amostras:

É possível agora, com todos os arquivos *.csv* criados, fazer uma analise rápida de uma amostra no LibreOffice Cal, no qual o Somatório dos números dão os 200 resultados.