

Lorsque l’exécutable est lancé, le main appelle la méthode Parse de la classe ArgumentParser qui vérifie que la commande d’appel est bien formée et qui analyse son contenue pour configurer la classe Analyse. La méthode renvoi alors un code (PARSE\_RESULTS).

La classe Analyse est la classe centrale de l’application, elle utilise LogReader pour récupérer les différents hits du fichier passer au lancement du programme, et construit les différentes structures de données en fonction de la configuration créé par ArgumentParser.

Une fois l’analyse terminée, on affiche un top 10 des ressources demandées et dans le cas ou l’utilisateur à demander un graph, on utilise GraphVizWritter et la méthode Write pour exporter la structure de données GraphMapper sous forme de graph.

Afin de ne pas avoir à faire de reconversion de nos données, nous avons exporter les adresses des différents strings de la structure en tant qu’ID dans le fichier .dot généré.

## Compilation et Exécution de Analog

## Compilation

* Se placer dans le dossier « Project »
* Exécuter « make init »
* Compiler un exécutable :
  + Version de production : Exécuter « make »
* Nettoyer les fichiers de build :
  + Fichiers .o : Exécuter « make clean »
  + Fichiers .o et exécutable : Exécuter « make clean-all »

**Exécution**

* Version de production/debug : Exécuter « bin /Analog » (Affichera le man)

## Man

Utilisation de ./analog [-e|-g|-t] <path>

<path> : chemin vers un fichier d’extension .txt ou .log.

-e : Exclut les documents de type image, css ou javascript.

-g <path.dot> : Genère le fichier <path.dot> au format GraphViz du graphe analysé.

-t [0-23] : Prend en compte que les hits dans le créneau horaire [t, t+1].