

Makefile

Makefile

- Objectif: apprendre à écrire un fichier Makefile lisible pour un projet de programmation, en tirant profit des dépendances implicites, de règles par défaut etc.

Compilation: Rappel

- Définition: *travail réalisé par un **compilateur** qui consiste à **transformer un code source lisible** par un humain en un **fichier binaire exécutable** par une machine.*
- Etapes de la compilation:
 - prétraitement par le pré-processeur
 - construction des fichiers objets
 - édition de lien (assemblage des fichiers objets pour créer l'exécutable)

Compilation: exemple simple

programme hello.cc

```
#include <iostream>
```

```
int main() {  
    std::cout << "Hello, world" << std::endl;  
}
```

- Compilation:

g++ -Wall -o hello hello.cc

- g++: compilateur
- -Wall: option pour afficher les warnings
- -o hello: spécifie le nom de l'exécutable généré
- hello.cc: fichier source

Compilation Séparée : Rappel

- Découpage de programme en plusieurs fichiers
 - Lisibilité, maintenance, réutilisabilité...
- Différents types de fichiers:
 - source (*.cc), contient le code des fonctions
 - fichier source "*principale*" contient la fonction *main*
 - en-tête (*.h), contient les déclarations des fonctions
- Construction de l'exécutable
 - fichiers objets (*.o), bibliothèque

Compilation Séparée: Rappel

- Plusieurs étapes:
 - on crée les fichiers objets à partir des fichiers sources

```
g++ -c Tableau.cc
g++ -c Main.cc
```
 - puis on les assemble pour créer l'exécutable

```
g++ Main.o Tableau.o -o testTab
```
 - on peut ensuite exécuter le programme

```
./testTab [arguments]
```

What does make make ?

- `make` est un utilitaire permettant:
 - de déterminer automatiquement les parties d'un programme à recompiler
 - d'exécuter les commandes appropriées
- Avantages de `make`
 - effectue les commandes en fonction des fichiers les plus récents
 - gère les dépendances
- S'appuie sur un fichier appelé ***Makefile***
 - il décrit l'organisation modulaire du programme et les règles de construction

Fichier Makefile

- La structure de base du Makefile est :

cible: dependances
commandes

...

- **cible**: nom du fichier créé
- **dependances**: liste des fichiers (ou règles) nécessaires à la construction de la cible
- **commandes**: commandes à effectuer pour créer la cible

Fichier Makefile

- La structure de base du Makefile est :

cible: dependances

■■■ commandes

...

- **cible**: nom du fichier créé
- **dependances**: liste des fichiers (ou règles) nécessaires à la construction de la cible
- **commandes**: commandes à effectuer pour créer la cible

ATTENTION les lignes de commandes sont précédées d'une tabulation (obligatoire!), et non d'espaces

Makefile: exemple simple

- Création de l'exécutable hello depuis le programme hello.cc
 - **cible:** hello
 - **dependance:** hello.cc
 - **commande:** g++ -Wall -o hello hello.cc

Makefile

hello: hello.cc

g++ -Wall -o hello hello.cc

- compilation: make
 - si le fichier hello est à jour (le fichier source n'est pas plus récent), la cible n'est pas reconstruite

Makefile: Cibles

- Un Makefile comporte souvent plusieurs cibles
 - `make cible` lance la fabrication de *cible*
 - `make` lance la fabrication de la première cible (hello)

Makefile

```
hello: hello.cc
    g++ -Wall -o hello hello.cc

hello.pdf: hello.cc Makefile
    a2ps -o - hello.cc Makefile | ps2pdf - hello.pdf

# Cibles habituelles
clean:
    rm -f *~ *.o

mrproper: clean
    rm -f hello
```

Compilation séparée: exemple

- Reprenons l'exemple d'AP1 qui comprend:
 - Tableau.cc, qui contient un ensemble de fonctions de traitement sur les tableaux
 - Tableau.h dans lequel les prototypes de ces fonctions sont définis
 - le programme Main.cc qui fait référence à ces fonctions
- La compilation devra donc comprendre
 - la compilation de Tableau.cc et de Main.cc pour fabriquer les modules objets relatifs
 - l'édition des liens des fichiers objets pour constituer l'exécutable testTab

Compilation séparée: exemple

Makefile

```
testTab: Tableau.o Main.o
    g++ -o testTab Main.o Tableau.o
```

```
Tableau.o: Tableau.cc Tableau.h
    g++ -Wall -c Tableau.cc
```

```
Main.o: Main.cc Tableau.h
    g++ -Wall -c Main.cc
```

```
code.pdf: Tableau.cc Tableau.h Main.cc
    a2ps -o - Tableau.cc Tableau.h Main.cc|ps2pdf - code.pdf
```

Cibles habituelles

```
clean:
    rm -f *~ *.o
```

```
mrproper: clean
    rm -f tabTest
```

Simplification

- Exemples de Makefile étudiés:
 - très simples, fonctionnent
 - limites :
 - un grand nombre de fichiers → un grand nombre de règles à définir
 - les fichiers peuvent être répartis dans plusieurs répertoires
 - changer de compilateur ou d'option → corriger toutes les commandes
- Écriture et entretien de Makefile pour de gros projets:
 - utilisation de variables, de dépendances et de règles implicites

Makefile: variables automatiques

- Contenu automatiquement défini
- $\$@$, contient la *cible*
testTab: Tableau.o Main.o
g++ -o $\$@$ Main.o Tableau.o
- $\$^$, contient la *liste des dépendances*
testTab: Tableau.o Main.o
g++ -o $\$@$ $\$^$
- $\$<$, contient la *première dépendance*
Tableau.o: Tableau.cc Tableau.h
g++ -Wall -c $\$<$

Main.o: Main.cc Tableau.h
g++ -Wall -c $\$<$

Makefile: variables \$@, \$^ et \$<

Makefile

```
testTab: Tableau.o Main.o
    g++ -o $@ $^
```

```
Tableau.o: Tableau.cc Tableau.h
    g++ -Wall -c $<
```

```
Main.o: Main.cc Tableau.h
    g++ -Wall -c $<
```

```
code.pdf: Tableau.cc Tableau.h Main.cc
    a2ps -o - $^ | ps2pdf - $@
```

Cibles habituelles

```
clean:
    rm -f *~ *.o
```

```
mrproper: clean
    rm -f testTab
```


Makefile: Variables

- Il est possible d'introduire d'autres variables
 - définies sous la forme `NOM = VALEUR`
 - valeur appelée avec `$(NOM)`
- Utilisées pour représenter tout ce qui peut évoluer:
 - listes de fichiers (sources, en-têtes, objets)
 - options de compilation
 - bibliothèques employées
 - ...

Exemple

Makefile

```
sources= Main.cc Tableau.cc  
entetes=Tableau.h  
objets=$(sources:.cc=.o)
```

```
testTab: $(objets)  
    g++ -o $@ $^
```

```
Tableau.o: Tableau.cc Tableau.h  
    g++ -Wall -c $<
```

```
Main.o: Main.cc Tableau.h  
    g++ -Wall -c $<
```

```
code.pdf: $(sources) $(entetes)  
    a2ps -o - $^ | ps2pdf - $@
```

```
# Cibles habituelles
```

```
clean:  
    rm -f *~ *.o
```

```
mrproper: clean  
    rm -f testTab
```

Commandes par défaut

- `make` possède des dépendances et règles par défaut
 - exp: `prog.cc` existe dans votre répertoire:
 - `make prog.o` lance `g++ -c -o prog.o prog.cc`
 - **dépendance implicite**
 - si `prog.cc` existe, le fichier `prog` dépend de `prog.cc`
 - **commande par défaut** pour fabriquer un exécutable
 - en l'absence de précision: utilisation de la commande de compilation adaptée
 - à partir d'un source C++ (nom + suffixe `.cc`) → **g++**

Commandes par défaut

Makefile

```
sources= Main.cc Tableau.cc  
entetes=Tableau.h  
objets=$(sources:.cc=.o)
```

```
testTab: $(objets)  
    g++ -o $@ $^
```

```
Tableau.o: Tableau.cc Tableau.h  
#    g++ -Wall -c $<
```

```
Main.o: Main.cc Tableau.h  
#    g++ -Wall -c $<
```

```
code.pdf: $(sources) $(entetes)  
    a2ps -o - $^ | ps2pdf - $@
```

Cibles habituelles

```
clean:  
    rm -f *~ *.o
```

```
mrproper: clean  
    rm -f testTab
```

Commandes par défaut

Makefile

```
sources= Main.cc Tableau.cc  
entetes=Tableau.h  
objets=$(sources:.cc=.o)
```

```
testTab: $(objets)  
    g++ -o $@ $^
```

```
Tableau.o: Tableau.cc Tableau.h  
#    g++ -Wall -c $<
```

```
Main.o: Main.cc Tableau.h  
#    g++ -Wall -c $<
```

```
code.pdf: $(sources) $(entetes)  
    a2ps -o - $^ | ps2pdf - $@
```

Cibles habituelles

```
clean:  
    rm -f *~ *.o
```

```
mrproper: clean  
    rm -f testTab
```

- **Pb:** perte de l'option **-Wall**
- **Sol:** utilisation de **variables** pour paramétrer la commande

Commandes par défaut

Makefile

CXXFLAGS=-Wall

sources= Main.cc Tableau.cc

entetes=Tableau.h

objets=\$(sources:.cc=.o)

testTab: \$(objets)
g++ -o \$@ \$^

Tableau.o: Tableau.cc Tableau.h
g++ -Wall -c \$<

Main.o: Main.cc Tableau.h
g++ -Wall -c \$<

code.pdf: \$(sources) \$(entetes)
a2ps -o - \$^ | ps2pdf - \$@

Cibles habituelles

clean:
rm -f *~ *.o

mrproper: clean
rm -f testTab

- **Pb:** perte de l'option **-Wall**
- **Sol:** utilisation de **variables** pour paramétrer la commande
- **CXXFLAGS** définit les options à passer au compilateur C++

Variables: conventions

- Pour les noms d'exécutables et d'arguments :
 - **CC** : compilateur C (gcc)
 - **CXX** : compilateur C++ (g++)
 - **CFLAGS** : paramètres à passer au compilateur C
 - **CXXFLAGS** : paramètres pour le compilateur C++
 - **LDFLAGS** : paramètres pour l'éditions de liens
 - ...
- Pour les noms de répertoires, les destinations:
 - prefix, bindir, libdir, includedir, mandir ...

Commandes par défaut

Makefile

CXXFLAGS=-Wall

sources= Main.cc Tableau.cc

entetes=Tableau.h

objets=\$(sources:.cc=.o)

testTab: \$(objets)
g++ -o \$@ \$^

Tableau.o: Tableau.cc Tableau.h
g++ -Wall -c \$<

Main.o: Main.cc Tableau.h
g++ -Wall -c \$<

code.pdf: \$(sources) \$(entetes)
a2ps -o - \$^ | ps2pdf - \$@

Cibles habituelles

clean:
rm -f *~ *.o

mrproper: clean
rm -f testTab

- **Cas de la cible testTab:**
 - La cible ne peut utiliser une règle par défaut que si son nom permet de retrouver les dépendances implicites.
 - Main dans notre cas

Commandes par défaut

Makefile

CXXFLAGS=-Wall

sources= Main.cc Tableau.cc

entetes=Tableau.h

objets=\$(sources:.cc=.o)

Main: \$(objets)
g++ -o \$@ \$^

Tableau.o: Tableau.cc Tableau.h
g++ -Wall -c \$<

Main.o: Main.cc Tableau.h
g++ -Wall -c \$<

code.pdf: \$(sources) \$(entetes)
a2ps -o - \$^ | ps2pdf - \$@

Cibles habituelles

clean:
rm -f *~ *.o

mrproper: clean
rm -f Main

- **Cas de la cible Main:**
 - la commande par défaut fait appel au compilateur C
gcc

Commandes par défaut

Makefile

CXXFLAGS=-Wall

sources= Main.cc Tableau.cc

entetes=Tableau.h

objets=\$(sources:.cc=.o)

%.o

\$(LINK.cc) -o \$@ \$^

Main: \$(objets)

g++ -o \$@ \$^

Tableau.o: Tableau.cc Tableau.h

g++ -Wall -c \$<

Main.o: Main.cc Tableau.h

g++ -Wall -c \$<

code.pdf: \$(sources) \$(entetes)

a2ps -o - \$^ | ps2pdf - \$@

Cibles habituelles

clean:

rm -f *~ *.o

mrproper: clean

rm -f Main

- **Cas de la cible Main:**

- Solution: **redéfinir la commande par défaut** utilisée pour créer un exécutable à partir des objets.

Construction automatique des dépendances

- `makedepend` effectue la recherche des dépendances entre les fichiers sources

- `makedepend Main.cc Tableau.cc`

→ ajoute à la fin du Makefile:

```
# DO NOT DELETE
```

```
Main.o: Tableau.h
```

```
Tableau.o: Tableau.h
```

- Dépendances obtenues en examinant les **inclusions** présentes dans les fichiers
 - inclusions éventuellement multi-niveaux
- Dépendances calculées se combinent avec les dépendances implicites
 - contrainte restante: indiquer comment **fabriquer les modules objets**

Construction automatique des dépendances

Makefile

```
CXXFLAGS=-Wall
```

```
sources= Main.cc Tableau.cc
```

```
entetes=Tableau.h
```

```
objets=$(sources:.cc=.o)
```

```
%.o:
```

```
$(LINK.cc) -o $@ $^
```

```
Main: $(objets)
```

```
code.pdf: $(sources) $(entetes)
```

```
a2ps -o - $^ | ps2pdf - $@
```

```
# Cibles habituelles
```

```
clean:
```

```
rm -f *~ *.o
```

```
mrproper: clean
```

```
rm -f Main
```

```
# Gestion automatique des dépendances
```

```
depend:
```

```
makedepend $(sources)
```

- make depend pour générer les dépendances:

→ avant la première utilisation

→ à chaque ajout de fichier source ou modification des inclusions

Construction automatique des dépendances

Makefile

CXXFLAGS=-Wall

sources= Main.cc Tableau.cc

entetes=Tableau.h

objets=\$(sources:.cc=.o)

%.o:

\$(LINK.cc) -o \$@ \$^

Main: \$(objets)

code.pdf: \$(sources) \$(entetes)

a2ps -o - \$^ | ps2pdf - \$@

Cibles habituelles

clean:

rm -f *~ *.o

mrproper: clean

rm -f Main

Gestion automatique des dépendances

depend:

makedepend \$(sources)

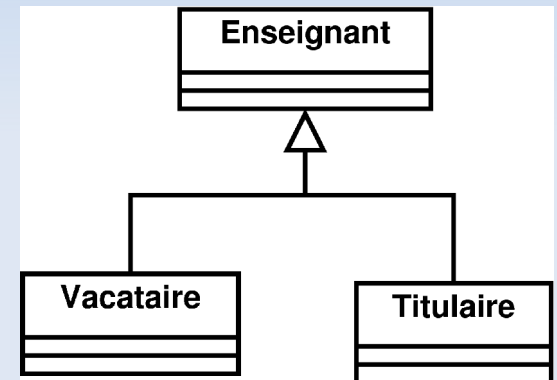
DO NOT DELETE

Main.o: Tableau.h

Tableau.o: Tableau.h

Exemple 1

- Projet de gestion des enseignements:
 - 3 classes: (un fichier source + un fichier en-tête)
 - Enseignant (abstraite)
 - Titulaire
 - Vacataire
 - 2 programmes:
 - paie.cc
 - emploiDuTemps.cc
 - 2 programmes de test:
 - testTitulaire.cc
 - testVacataire.cc



Exemple 1

- Dépendances explicites des programmes:
 - **paie**: paie.o Enseignant.o Titulaire.o Vacataire.o
 - **emploiDuTemps**: emploiDuTemps.o Enseignant.o Titulaire.o Vacataire.o
 - **testTitulaire**: testTitulaire.o Enseignant.o Titulaire.o
 - **testVacataire**: testVacataire.o Enseignant.o Vacataire.o
- 10 sources, 7 modules objets, 4 executables

Exemple 1

Makefile

CXXFLAGS=-Wall

progs = paie.cc emploiDuTemps.cc

tests = testTitulaire.cc testVacataire.cc

classes = Enseignant.cc Titulaire.cc Vacataire.cc

entetes = \$(classes:.cc=.h)

execs = \$(tests:.cc) \$(progs:.cc=)

all: \$(execs)

paie: paie.o Enseignant.o Titulaire.o Vacataire.o

emploiDuTemps: emploiDuTemps.o Enseignant.o
◀Titulaire.o Vacataire.o

testTitulaire: testTitulaire.o Enseignant.o Titulaire.o

testVacataire: testVacataire.o Enseignant.o Vacataire.o

listing.pdf: \$(applis) \$(entetes) \$(classes) Makefile
a2ps -o - \$^ | ps2pdf - @<

listing-tests.pdf: \$(tests) \$(entetes) \$(classes) Makefile

a2ps -o - \$^ | ps2pdf - @<

#####

%.o

\$(LINK.cc) -o \$@ \$^

clean:

rm -f *~ *.o *.bak

mrproper: clean

rm -f \$(execs)

depend:

makedepend \$(progs) \$(classes)

Exemple 2

- L'exemple inclu:
 - une bibliothèque: MyLib
 - un programme projet.cc qui utilise MyLib
- 3 Makefile en charge:
 - (1) de la création de l'exécutable du projet
 - (2) au niveau de l'arborescence de MyLib
 - (3) de la création de la bibliothèque depuis les sources

Exemple 2	
Projet	MyLib
-- <u>Makefile</u> (1)	-- <u>Makefile</u> (2)
-- projet.cc	-- include
	-- mylib.h
	-- lib
	-- src
	-- <u>Makefile</u> (3)
	-- carre.cc
	-- cube.cc

Bibliothèque

- Rappel: une **bibliothèque** (librairie) est un **ensemble de fonctions utiles**, regroupées et mises à disposition.
 - fonctions regroupées par appartenance à un domaine: mathématique, graphique, tris, ...
 - pas de fonction principale, ne peut être exécutée directement
 - contient du code utile, évite de le réécrire à chaque fois
 - statique (.a) ou dynamique (.so ou .ddl)
 - intervient dans la création de l'exécutable au moment de **l'édition de liens**
 - utilisation nécessite d'**inclure les en-têtes**

Arbre de Projet: convention

- Organisation usuelle de répertoires dédiés:
 - **bin**: fichiers binaires
 - **include**: fichiers en-têtes
 - **lib**: bibliothèques
 - **src**: sources
 - **test**: programmes de test
- Projet peut contenir différents modules, donc plusieurs sous-arbres

Exemple 2	
Projet	MyLib
-- <u>Makefile</u> (1)	-- <u>Makefile</u> (2)
-- projet.cc	-- include
	-- mylib.h
	-- lib
	-- src
	-- <u>Makefile</u> (3)
	-- carre.cc
	-- cube.cc

- Plusieurs Makefile
 - multi-niveaux

Exemple 2: installation de la bibliothèque

Makefile (2)

```
#  
# Usage récursif de make  
#  
all clean mrproper :  
    $(MAKE) -C src $@
```

Exemple 2

Exemple 2	
Projet	MyLib
-- <u>Makefile</u> (1)	-- <u>Makefile</u> (2)
-- projet.cc	-- include
	-- mylib.h
	-- lib
	-- src
	-- <u>Makefile</u> (3)
	-- carre.cc
	-- cube.cc

Exemple 2: installation de la bibliothèque

Makefile (3)

```
### Fabrication d'une bibliothèque statique
### à partir de deux modules objet.
sources = carre.cc cube.cc
objects = $(sources:.cc=.o)

# bibliothèque à fabriquer (commence par lib)
library = ../lib/libmylib.a

# le répertoire avec le .h de la bibliothèque
include_dir = ../include
CPPFLAGS = -I$(include_dir)

CXXFLAGS = -Wall
#####

all: $(library)

$(library): $(objects)
    ar rcs $@ $^

# objets fabriqués selon les règles implicites
#####
depend:
    makedepend -I$(include_dir) $(sources)

clean:
    rm -f *~ *.bak \#*
```

Exemple 2

Projet	MyLib
-- <u>Makefile</u> (1)	-- <u>Makefile</u> (2)
-- projet.cc	-- include
	-- mylib.h
	-- lib
	-- src
	-- <u>Makefile</u> (3)
	-- carre.cc
	-- cube.cc

```
mrproper: clean
    rm -f *.o $(library)
```

```
#####
# DO NOT DELETE
```

```
carre.o: ../include/mylib.h
cube.o: ../include/mylib.h
```

Exemple 2: installation de la bibliothèque

Makefile (1)

```
### Utilisation d'une bibliothèque
source = projet.cc
bin    = $(source:.cc=)

# le répertoire qui contient les entetes
CPPFLAGS = -I../MyLib/include

# repertoire et nom de la bibliothèque
# (sans le préfixe lib ni le suffixe .a)
LDLIBS   = -L ../MyLib/lib -lmylib

CXXFLAGS = -Wall -pedantic

#####
all: $(bin)

%: %.o
    $(LINK.cc) -o $@ $^ $(LDLIBS)

clean:
    rm -f *.o *~

depend:
    makedepend $(CPPFLAGS) $(source)
```

Exemple 2

Projet	MyLib
-- <u>Makefile</u> (1)	-- <u>Makefile</u> (2)
-- projet.cc	-- include
	-- mylib.h
	-- lib
	-- src
	-- <u>Makefile</u> (3)
	-- carre.cc
	-- cube.cc

```
#####
# DO NOT DELETE
```

```
projet.o: ../MyLib/include/mylib.h
```

Tutoriels

- Nombreux tutoriels disponible:
 - exp: http://www.siteduzero.com/tutoriel-3-31992-compilez-sous-gnu-linux.html#ss_part_3