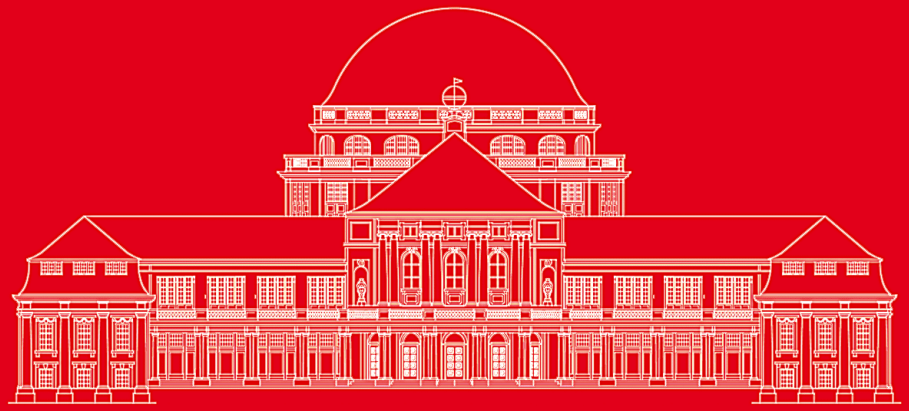
Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Praktikum: Paralleles Programmieren für Geowissenschaftler

Prof. Thomas Ludwig, Hermann Lenhart



Dr. Hermann-J. Lenhart
hermann.lenhart@zmaw.de

Makefile

- Einführung
- Regeln
- Praxis mit Beispielen

Einführung Makefile I

make ist ein Tool, mit dem komplexe Programme auf einfache Weise kompiliert und ausgeführt werden können.

make liest ein sogenanntes *Makefile* (oder *makefile*),
in dem die Abhängigkeiten des Übersetzungsprozesses von Programmen formalisiert erfasst sind.

D.h. der Ablauf von Quelldateien -> Objektdateien => Ergebnissen
wird über das *Makefile* gesteuert.

Einführung Makefile II

Das *Makefile* wird aufgerufen durch den Befehl

- **make**

Wird in einem umfangreichen Programmcode
z.B nur eine Datei geändert,
so kann durch den Aufruf von **make** gezielt dieses überarbeitete Teil
in ein neues lauffähiges Programm eingebunden werden.

Dies geschieht indem das Programm „make“ die Zeitstempel der Dateien auswertet.

Einführung Makefile III

Bei der Steuerung vom Ablauf

Quelldateien -> Objektdateien => Ergebnissen

müssen Abhängigkeiten von Dateien berücksichtigt werden,

die eine bestimmte Reihenfolge im Kompilieren und Linken erfordern.

Das Makefile kann darüber hinaus noch weitere Steuerfunktionen für das Programm übernehmen, z.B: Aufruf von MPI oder OpenMP,

sowie Befehle auf der Commandozeile.

Makefile Regeln

Ein *Makefile* hat eine fest vorgeschriebene Struktur:

Target …… : Voraussetzung …….

Kommando

z.B. für das „Hallo World“ Program:

```
hello.x:  hello.f90
          f95 –o hello.x  hello.o
```

Makefile Regeln

Das wichtigste Markmal vom *Makefile* **ist unsichtbar:**

⟼ *der Tab!*

Die korrekte Syntax lautet:

```
hello.x:  hello.f90
⟼       f95 –o hello.x  hello.o
```

Makefile Regeln

D.h. die korrekte Syntax für ein *Makefile* lautet:

Target …… : Vorraussetzung …….

↦ *Kommando*

Der Tab kann nicht durch die entsprechende Anzahl von Leerzeichen ersetzt werden, was optisch erst mal gleich aussieht!

Dies führt häufig zu der Fehlermeldung: *** missing separator

Makefile Regeln

D.h. die korrekte Syntax für ein *Makefile* lautet:

Target …… : Vorraussetzung …….

↦ *Kommando*

Der Tab kann nicht durch die entsprechende Anzahl von Leerzeichen ersetzt werden, was optisch erst mal gleich aussieht!

Dies führt häufig zu der Fehlermeldung: *** missing separator

Praxis Makefile

Der Gebrauch vom *Makefile* erleichtert die Überarbeitung von Programmen, indem nur die aktuellen Änderungen neu kompiliert werden.

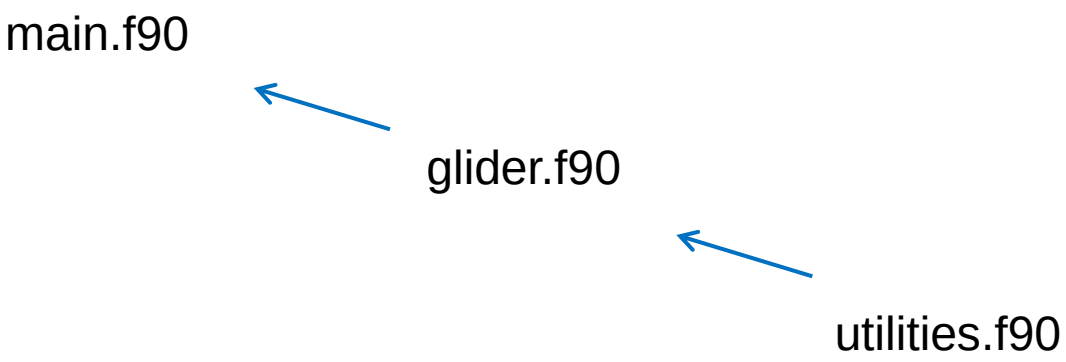
Bei größeren Projekten spart dies Zeit beim Testen der Programme.

Wichtig dabei ist dass alle Abhängigkeiten (Dependencies) im *Makefile* abgebildet wurden.

Dazu wird ein „Dendency Tree“ oder „Abhänigigkeitsbaum“ ausgewertet.

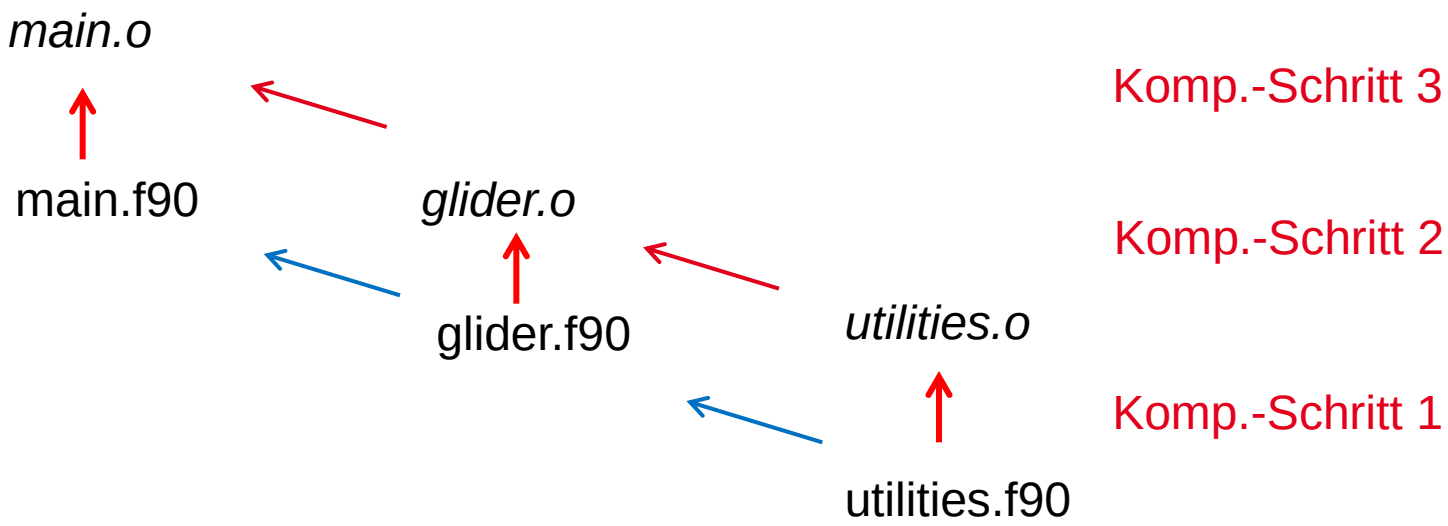
Praxis Makefile: Darstellung Glider Programm

Einfache Abhängigkeit der Programme untereinander



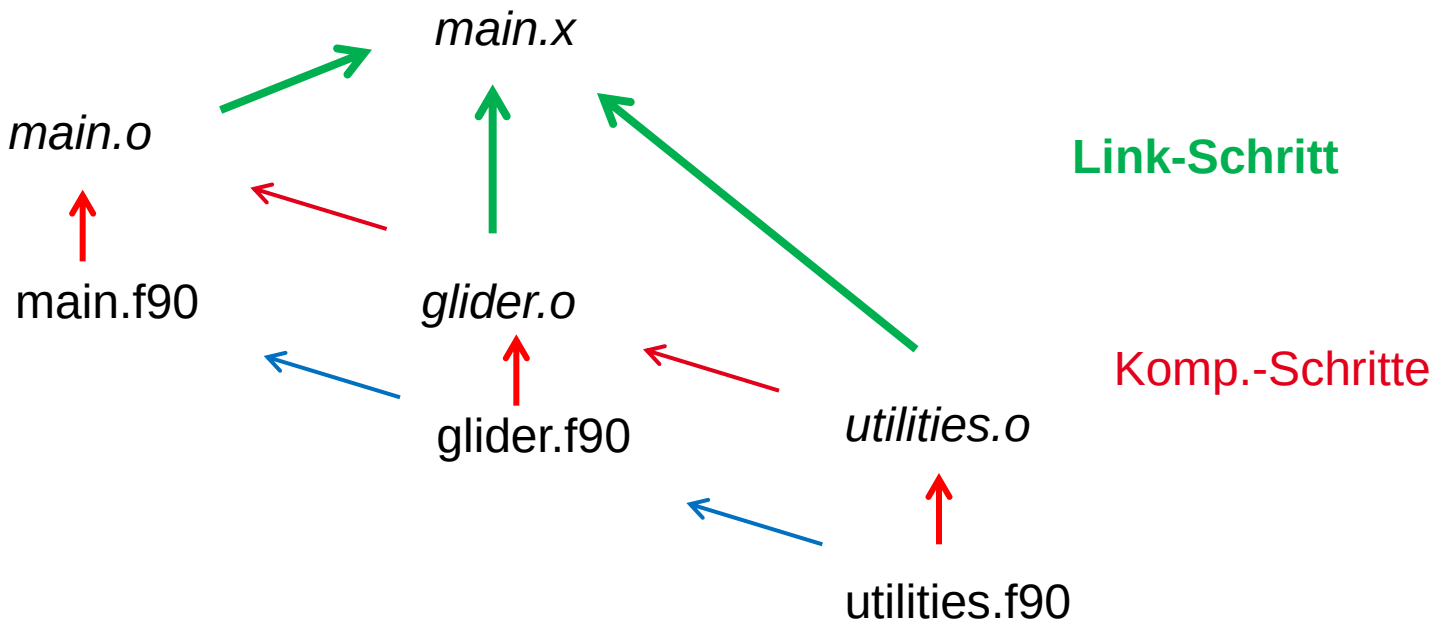
Praxis Makefile: Darstellung Glider Programm

Aus Source Files werden über das Kompilieren Objektdateien erstellt.



Jedes neu generierte File entspricht einer *Dependency*, d.h. ein Schritt im *Makefile*.

Praxis Makefile: Darstellung Glider Programm



Der Link-Schritt entspriche ebenfalls einer *Dependency* im *Makefile.*

Das Makefile für Glider

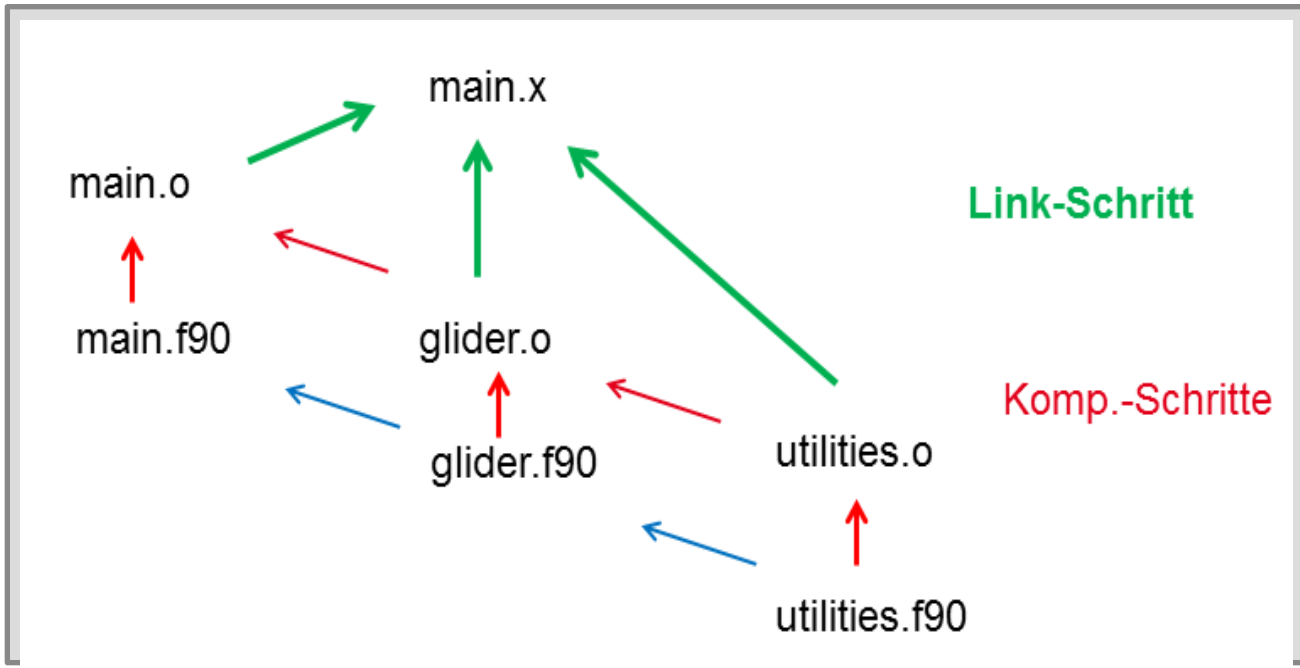
```
main.x: main.f90 glider.o utilities.o
        f95 -o main.x main.f90 glider.o  utilities.o

utilities.o: utilities.f90
        f95 -c utilities.f90

glider.o: glider.f90 utilities.o
        f95 -c glider.f90

run:main.x
        ./main.x

clean:
        rm *.o main.x
```



Danke,
gibt es noch Fragen?