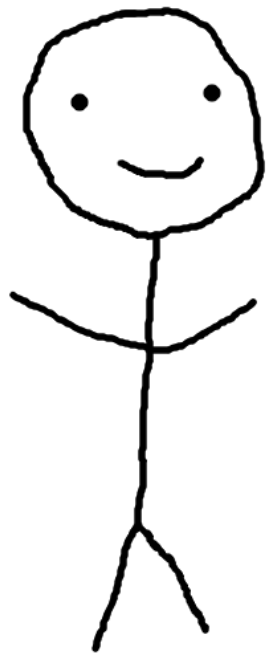
# Make

## Or How I Learned To Stop Worrying And Love The Compilation Process

Deniz Koluaçık

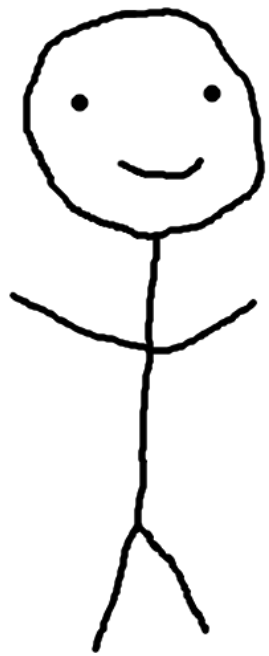
8 Mart 2020



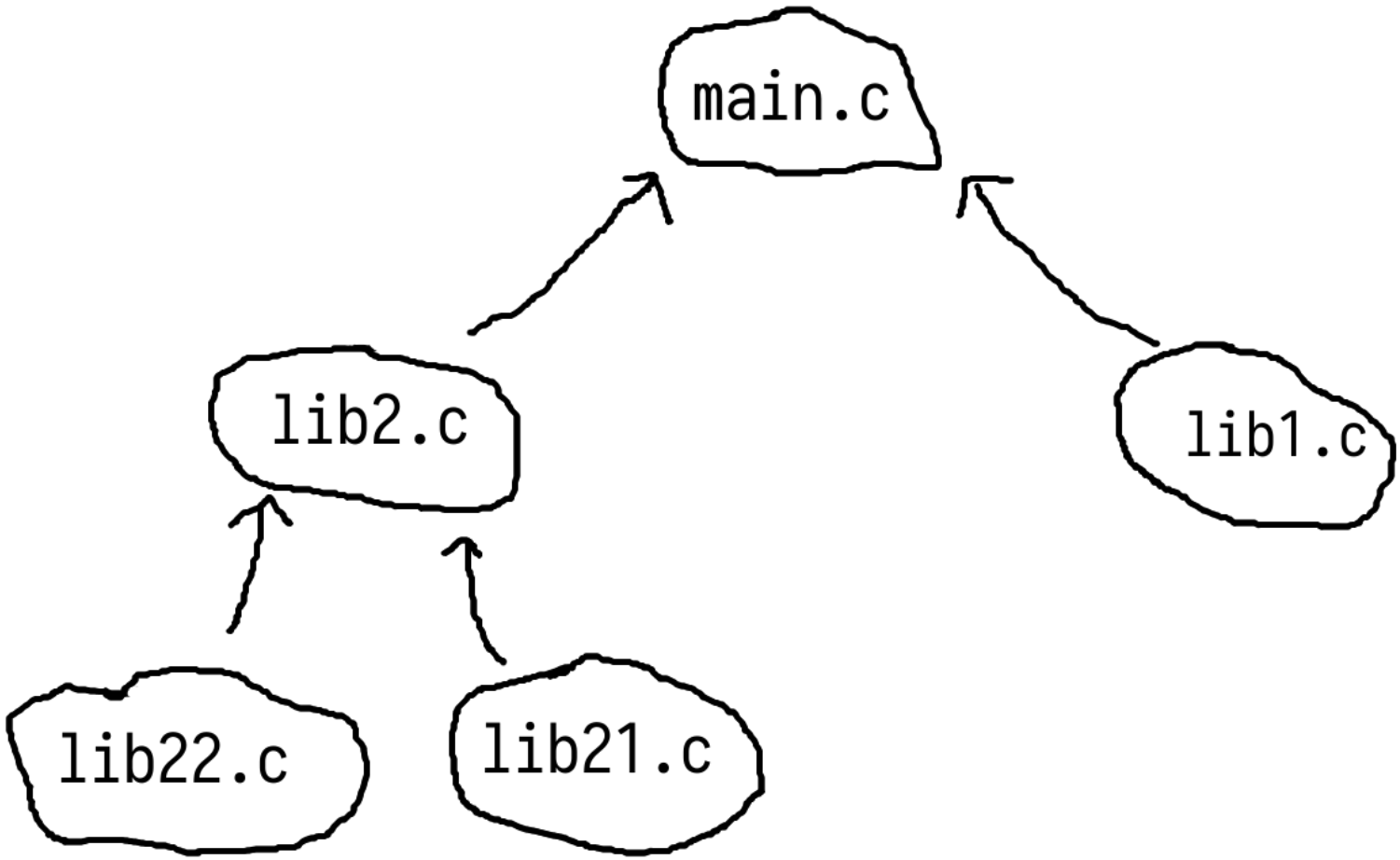
# İçindekiler



# Neden Make?



1




[1] Aslında dosyaların derlenebilmesi için böyle bir gereklilik ağacı mevcut değil. Yani diğer .c dosyalarının derleme esnasında bir arada olması gerekmiyor, bu gereklilik linkleme anında açığa çıkıyor. Buradaki grafik programın parçaları arasındaki mantıksal gereklilik ağacı.

# Naif Yöntem

```
gcc -o main main.c lib2.c lib21.c lib22.c lib1.c
```

Sıkıntılar:

- Ufak değişikliklerde bütün dosyalar tekrar derlenir, zaman ve enerji...
- Dosya sayısı arttıkça karmaşıklaşır.

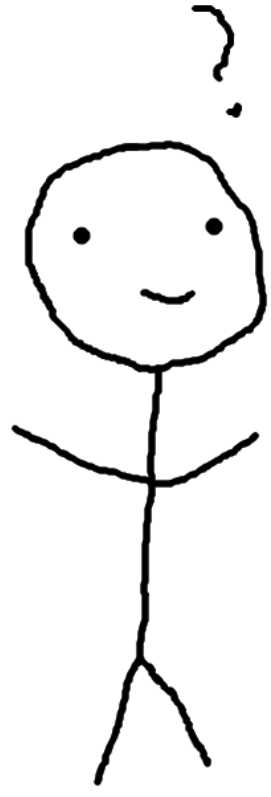
# Başka Bir Naif Yöntem

```
gcc -c -o lib21.o lib21.c
gcc -c -o lib22.o lib22.c
gcc -c -o lib2.o lib2.c
gcc -c -o lib1.o lib1.c
gcc -c -o main.o main.c
gcc -o main main.o lib1.o lib21.o lib22.o lib2.o
```

Sıkıntılar:

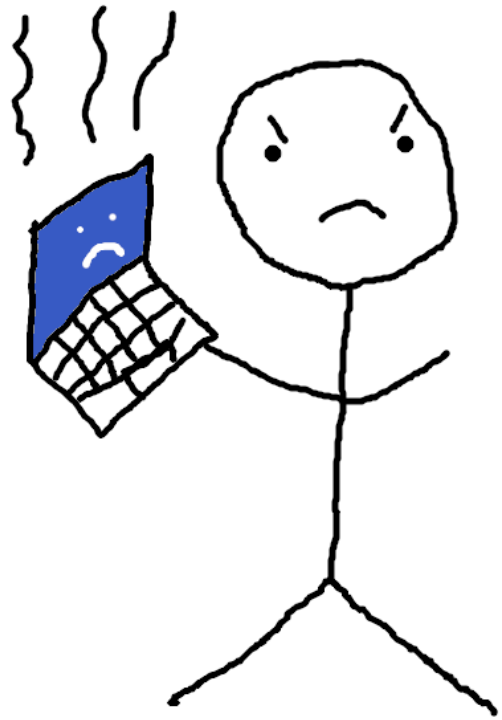
- Kullanıcı hangi komutların gerekli olduğunu düşünmelidir[2].
- Fazlasıyla karmaşık.

[2] Asıl sıkıntı header dosyalarında bir değişiklik yapıldığında başlıyor.



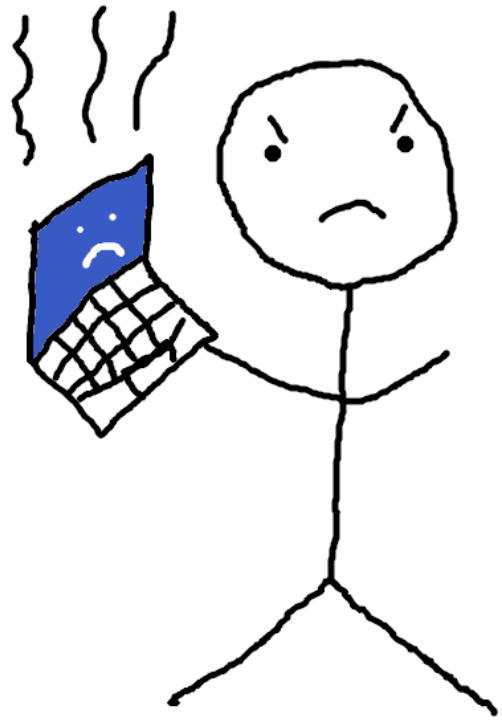
# Linux 5.5 70,000'den Fazla Dosyadan Oluşuyor

Soru: Derleme işlemini daha kolay yapmanın bir yolu var mı?

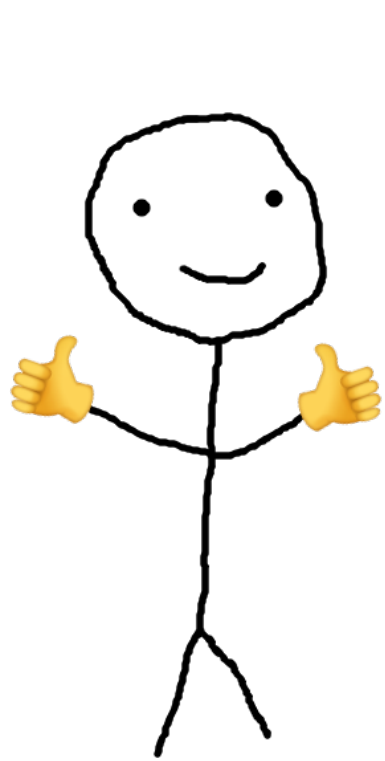


# Make!

1976 Stuart Feldman (Bell Labs)
1988 GNU Project



# MAKE100: Make Nasıl Çalışır?



- Çalışma Mantığı
- Temel Syntax

# Make Dosyasının İçeriği

```
target: prerequisites
    recipe

target: prerequisites
    recipe

target: prerequisites
    recipe
```

(Her biri bir kural.)

# Örnek

```
main: main.o lib1.o
      gcc -o main main.o lib1.o

main.o: main.c lib1.h
      gcc -c -o main.o main.c

lib1.o: lib1.h lib1.c
      gcc -c -o lib1.o lib1.c
```

# Target, Recipe, Prerequisite

# Target'ı (Yeniden) Oluşturmalı Mı?

Herhangi bir target için:

- Prerequisite bir Target'sa onu oluşturmalı mı?[3]
- Target var mı, varsa güncel mi? (son değiştirilme)
  - Hayır: **Target'ı oluştur.**
  - Evet: **Target güncel.**

[3] Recursion!

```
main: main.o lib1.o
	gcc -o main main.o lib1.o

main.o: main.c lib1.h
	gcc -c -o main.o main.c

lib1.o: lib1.h lib1.c
	gcc -c -o lib1.o lib1.c
```

# Syntax

```
main: main.o lib1.o
      gcc -o main main.o lib1.o
```

```
main: main.o lib1.o
      gcc -o main main.o lib1.o
      echo "Iyi günler!"
```

# Variables

```
VAR = main
$(VAR): main.o lib1.o...
```

# Variables

```
VAR = main
$(VAR): main.o lib1.o...

gcc -Wall -ansi -pedantic-errors -g -O0 -o main
main.o lib1.o
```

yerine...

```
FLAGS = -Wall -ansi -pedantic-errors -g -O0
gcc $(FLAGS) -o main main.o lib1.o
```

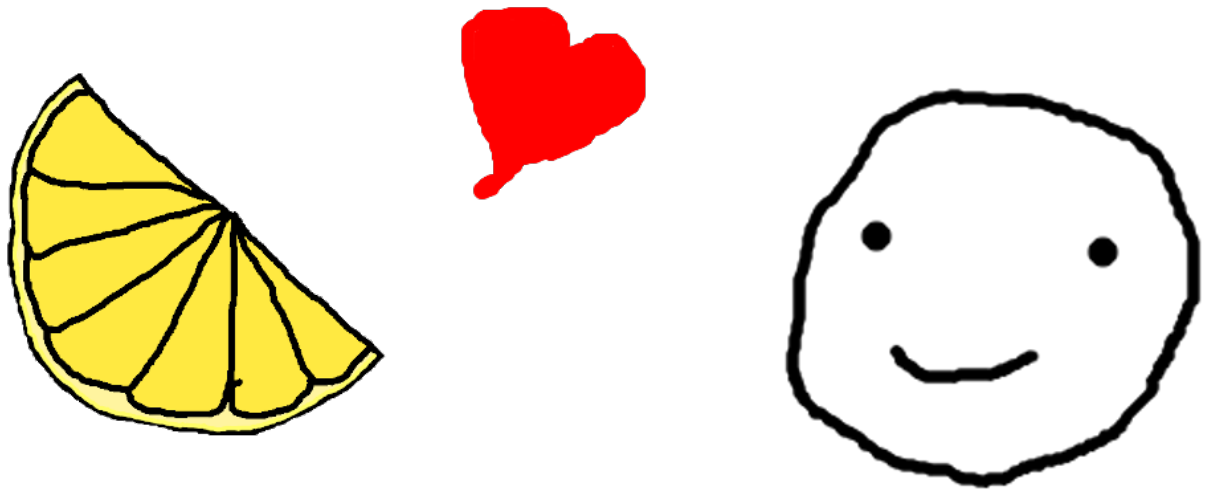
```
OBJECTS = main.o lib1.o lib21.o lib22.o lib2.o
main: $(OBJECTS)
	gcc -o main $(OBJECTS)
```

# Wildcards

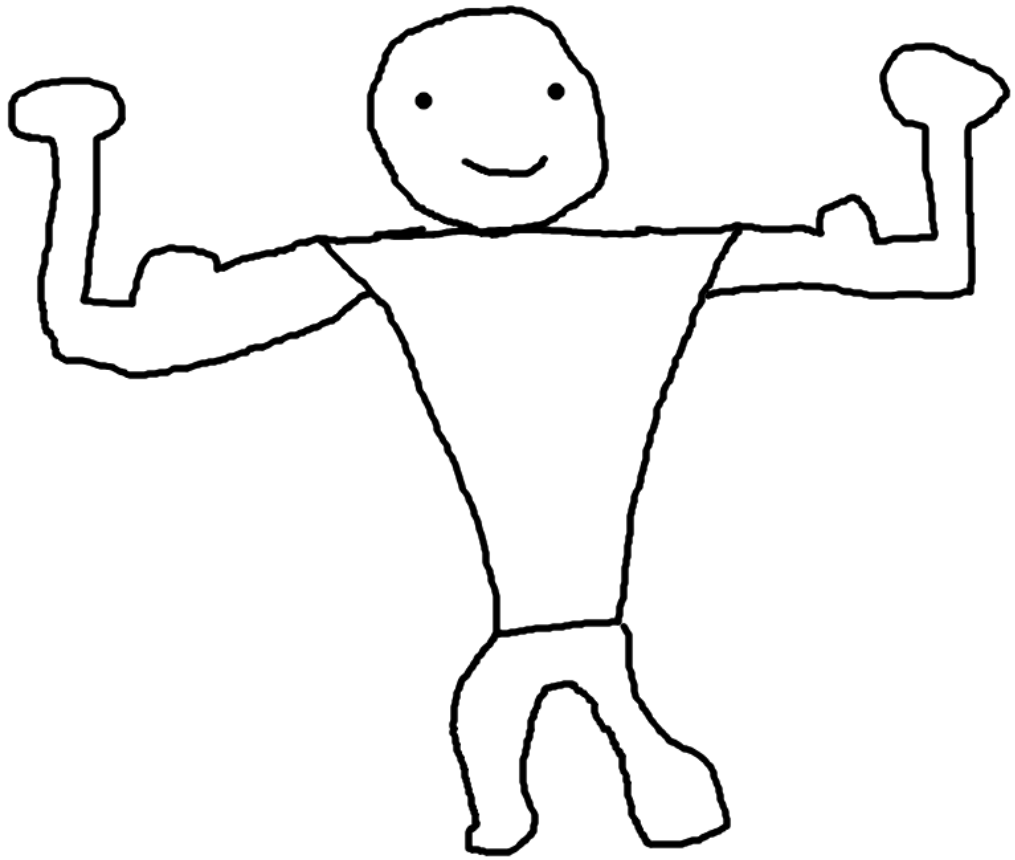
`*.o`: sonu “`.o`” ile biten bütün dosyalar.

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

# Artık Kendi Makefile'ımızı Yazabiliriz!



# MAKE200: Güçlü Özellikler!



# Ignoring errors

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

Peki ya “main” dosyası o an yoksa?

# Ignoring Errors

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

Başına "-" konan satırlarda karşılaşılan hatalar tarifi durdurmaz.

# Phony Targets

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

Peki ya dizinde “clean” isimli bir dosya varsa?

```
Make: ‘clean’ is up to date.
```

# Phony Targets

```
.PHONY: clean[4]
clean:
	-rm main
	-rm *.o
```

Dizinde ‘clean’ isimli bir dosya olsa bile bu kural düzgün çalışır.

[4] .PHONY sadece özel adı olan bir kuraldır.

# Recipe Echoing

```
.PHONY: clean
clean:
	-@rm main
	-@rm *.o
```

Make tarifte yer alan bütün komutları yazdırır.
Bunun önüne geçmek için satırların önüne '@' konulabilir.

Özellikle `echo` komutlarının başına yazmak çok iyi bir fikir.

# Implicit Variables

$(CC): C Compiler, cc (cc → gcc, symlink[5])
$(CXX): C++ Compiler, g++
$(CFLAGS): C flags, empty string
$(CXXFLAGS): C++ flags, empty string

$(CPPFLAGS): C Preprocessor flags, empty string.

[5] ls -l /usr/bin/cc

# Implicit Rules

make yaygın bazı kuralları otomatik olarak oluşturur.

```
foo.o: foo.c
	$(CC) $(CPPFLAGS) $(CFLAGS) -c -o foo.o foo.c
```

Benzeri cpp için de mevcut. (CXX, CXXFLAGS)

- Implicit rule'ların çalışması için dosyada o Target ve Prerequisite'lerin varlığından söz etmeye gerek yoktur.
  `make lib22.o`
- Yine de istersek bir bir Target'a elle Prerequisite belirleyerek Implicit rule'a katkıda bulunabiliriz.
  `lib1.o: lib1.c lib.h`
- Implicit rule'lar zincirleme çalışabilir. Bir prerequisite'in implicit bir tarifi olabilir.

# Implicit Linking Rule

.c(pp) dosyalarını derlemek için olduğu gibi .o dosyalarını linklemek için de bir Implicit rule mevcut.

```
n: n.o
	$(CC) $(LDFLAGS) n.o $(LOADLIBES) $(LDLIBS)
```

Birden fazla .o dosyasını linklemek için Target ve Prerequisite belirtmeliyiz.

```
main: main.o lib1.o lib2.o lib21.o lib22.o
```

# Pattern Rules

```
lib1.o: lib1.h lib1.c
lib2.o: lib2.h lib2.c
lib21.o: lib21.h lib21.c
lib22.o: lib22.h lib22.c
```

Genelleştirilebilir.

```
%.o: %.c %.h
```

%, make xxx.o formatında girilen herhangi bir komutta xxx'in değerini alır.
%, herhangi bir prefix ve suffixle tanımlanabilir. (of%off gibi)
%, empty string olamaz.
%'nun değerini aldığı string'e *stem* denir.

# Automatic Variables

Bir kuralın çalıştırılması anında değer alan değişkenlerdir.
Pattern rule'larda çok faydalıdır.

$@: Target'ın ismi.

```
he%he:
        @echo $@
```
(make hehehe, stdout'a hehehe yazdırır.)

# Başlıca Automatic Variable'lar

Çok sayıda tanımlı automatic variable var. Başlıcaları şunlar:

- $@ : Target adı
- $< : İlk prerequisite
- $^ : Bütün preprequisite'ler
- $% : Stem

```
he%he:
        @echo $*
```
(make hehehe, stdout'a he yazdırır.)

# Gerçekçi Bir Örnek

```
%.o: %.c %.h
	@echo "making $@!"
	$(CC) $(CPPFLAGS) $(CFLAGS) -c -o $@ $<
```

Sorun: Implicit rule'ların dışında bir genel kural istiyorum, ama bunu pattern matching kadar genel bir şekilde değil, birkaç dosya için yapmak istiyorum.

# Static Pattern Rules!

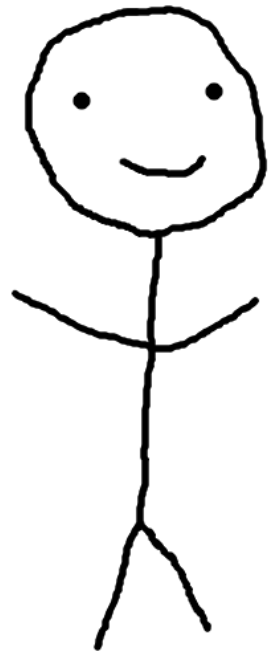
Pattern rules, ama belli hedefler için.

```
Targets: Pattern: Prerequisites
      Recipes...
```

```
lib1.o lib2.o lib21.o lib22.o : %.o: %.c %.h
	$(CC) $(CFLAGS) -c -o $@ $<
```

Targets kısmı için variable kullanabiliriz.

```
LIBOBJ = lib1.o lib2.o lib21.o lib22.o
$(LIBOBJ): %.o: %.c %.h
	$(CC) $(CFLAGS) -c -o $@ $<
```



# Şimdi Nereye?

```
$ info make
```

# Dinlediğiniz İçin Teşekkürler