Makefile

Makefile neden var?

Tekrar tekrar derlenmesi gereken büyük projelerde gerekli her şeyin tek seferde derlenmesi için ortaya çıktı. Özellikle C ve C++ dillerinde yaygın olarak kullanılıyor. Diğer dillerde olmamasının sebebi ise onların kendilerine ait bir derleme düzenleme araçlarının olmasıdır. Bu durumu kullanıcıya bırakmak istememişlerdir. Bu sebeple otomatik olarak gerçekleşmektedir lakin makefile eski olduğu için manuel olarak en başta girilmesi gerekmektedir lakin makefile bundan daha da fazlasıdır.

Makefile'a alternatif var mı?

C ve C++ için SCons, Cmake, Bazel ve ninja olmakla birlikte Java için Ant, Maven ve Gradle bulunmaktadır.

Makefile hedefi nedir?

Aslında oldukça basit. Üzerinde değişiklik yapılan dosyaları yeniden derlemek. Bütün dosyaların derlenmesi için de yollar mevcut elbette.

Makefile yapısı:

Makefile dosyaların değiştiğini nasıl anlar?

Makefile yapısal olarak dosyaların tarihlerine önem verir. Bir dosya başka bir dosyaya ihtiyaç duyuyor ve ihtiyaç duyduğu dosya kendisinden sonra oluşturulmuşsa o dosyayı derler. Eğer dosya kendisinden önce oluşturulmuşsa o dosyayı yeniden derlemeye gerek duymaz. Dosyada değişiklik olup olmadığını bu şekilde anlar.

Makefile içindeki kurallar nedir?

Bir makefile'ı makefile yapan şey kurallardır. Burada dikkat edilmesi gereken

şey sadece ilk kural çalıştırılır ve kural içindeki kodlar bir Tab konarak yapılır. Buna dikkat edilmesi gerek yoksa makefile hata verebilir.

```
Merhaba:
    echo “Merhaba”
    echo “Nasılsın”
```

Bağımlılık nedir?

Az önce bahsettiğim ihtiyaç şu şekilde olmaktadır.

```
Nasılsın: Merhaba
    echo “Nasılsın”
Merhaba:
    echo “Merhaba”
```

Yukarıdaki örnekte Nasılsın kuralı Merhaba kuralına bağlıdır. Yani Merhaba kuralı tamamlanmadan nasılsın kuralı çalıştırılamaz. Bu sebeple çıktı “Merhaba Nasılsın” olacaktır.

Clean yapısı:

Clean sıklıkla oluşturulan dosyaları silmek için kullanılır. Buna örnek olarak .o dosyaları verilebilir. Bu dosyalar sonradan oluşturulur ve belli durumlarda silinmesi pratik olabilir. Bu durumlar için bir clean dosyası yazılır ve o kullanılır.

```
dosya:
    touch dosya
clean:
    rm -f dosya
```

Burada dikkat edilmesi gereken iki durum var. Birincisi clean kuralı başka hiçbir kurala bağımlı değil bu sebeple özellikle çalıştırılmadığı sürece clean kuralı çalışmaz. Bu sebeple clean çalıştırmak için `make clean` kullanılır. Diğer bir durum ise kuralı çalıştırırken eğer clean adında bir dosyamız varsa kuralın çalışmayacağıdır. Bu durumu ileride anlatacağım .PHONNY ile düzelteceğiz.

Makefile’da değişkenler.

Makefile kendi başına küçük bir dil sayılabilir ve her dilde olduğu gibi değişkenler burada da mevcuttur. Değişkenler sadece string olabilir. Genel olarak := kullanılır ama = da geçerlidir.

```
dosyalar := dosya1 dosya2
```

Bir veya daha fazla alıntının bir anlamı yoktur.

```
a := bir ki
b := ‘bir ki’
```

İlkinin anlamı “bir ki” olmasına rağmen ikincisinin anlamı “ ‘bir ki’ ” dir. Yani ne yazdıysak o.

Değişkenler 3 şekilde kullanılabilir.

```
x: kas adam
all:
    echo $(x)
    echo ${x}
    echo $x
```

Her ne kadar çalışsa da $x şeklinde yazmak tavsiye edilmez.

Birden fazla kuralım var çalıştırmanın bir yolu yok mu?

Tabikide bunu yapmanın da yolu var. Kuralları tek seferde veya sıralı şekilde birbirlerine bağlamak işe yarayacaktır.

```
all: bir iki uc

bir:
    echo “bir”
iki:
    echo “iki”
uc:
```

```
    echo “uc”
```

Peki bunu yapmanın otomatik yolu yok mu?

Elimizde birkaç yol mevcut.

```
all: f1.o f2.o

f1.o f2.o:
    echo $@
```

$@ işareti kural adındaki değerleri otomatik olarak yerleştirir.

Yok mu daha hoj bir şeyler?

*Wildcard dosya sistemini aramaya ve tanımlı olanları getirmeye yarar.

```
print: $(wildcard *.c)
```

Bu şekilde kullanılır. * işareti direk olarak dosyalarda kullanılmamalıdır ve * ile eşleşen dosya bulamazsa olduğu gibi kalır.

Değiştirme fonksiyonu

% ile değerleri değiştirebilir. Bunu wildcard ile birlikte kullanıp gelen değerleri değiştirip dosyalar oluşturabiliriz.

Bu kodlar rahatsız ediyor kapatamaz mıyız?

Kodun başına eğer @ getirilirse o kod terminalde gözükmez.

Satırları bağlamanın yolu yok mu?

Her satır farklı bir shellde çalışıyor gibi olur bu sebeple sıradaki satır ile öncekini kıyaslayamayız çünkü ayrı shelldeler gibi davranıyor \ işareti koyduğumuzda ise aynı yerde çalışıyorlar.

Kod içinde $ işaretini yazdırmak.

Bunun için iki adet $ yazıyoruz.

Hata düzeltmeye yardımcı olan şeyler:

Hata düzeltmek programcılığın en önemli kısımlarından birisi ve makefile bunun için bize yardımcı birkaç özelliğe sahip.

- komutu hataları susturmaya yarar.

-i komutu bütün hataları susturmaya yarar.

-k komutu hata ile karşılaştığında compile etmeyi bırakmamasını sağlar. Bu sayede bütün satırlardaki hataları görebilirsiniz.

Recursive make nasıl yapılır.

Normal make yerine $(MAKE) yazıyoruz ve Recursive makefileımız hazır oluyor.

Makefilelar arasında veri gönderimi nasıl yapılır?

Export komutu makefile içindeki verilerin dışarı gönderilmesini sağlar ve içinde çalıştırdığımız bütün makeler için geçerlidir.

Değişken tanımlarken = ve := söylendi peki ya farkları neler?

= atama içindeki bütün değişkenleri genişletir yani yerine yazar. := ise o zamana kadar tanımlı olanları yazar tanımlı olmayanları yazmaz

```
#Değişken içinde değişken var
bir = bir ${sonradan-tanımlanacak}
#Sonradan tanımlandığı için yazılmayacak
```

```
two:= iki ${sonradan_tanımlanacak}

sonradan_tanımlanacak = sonrası geldi

all:
    echo $(bir)
    echo $(iki)
```

Buranın çıktısında ikinin olduğu yerde “sonrası geldi” yazmayacak çünkü := konulduğunda o değer tanımlanmamıştı.

```
bir = onceden tanımlı
#bir değeri önden tanımlanıyor bu yüzden aşağıdaki değer yazdırıldığında birin içindeki yazı da yazacak
bir := ${bir} gordun mu

all.
    echo $(bir)
```

Buranın çıktısında ise “onceden tanımlı”

yazısı gelecek çünkü en başta tanımlandı.

Default değer atamak mümkün mü?

Varsayılan olarak değer atamak istiyorsak ?= kullanımı yeterli olacaktır. Eğer değişken tanımlı değilse onun içindeki değer kullanılacaktır.

```
bir = sa
bir ?= atandığından bir anlamı yok
iki ?= başka iki olmadığından bu yazacak

all:
    echo $(bir)
    echo $(iki)
```

Tırnak kullanmıyoruz madem boşluk nasıl atılır?

Tırnak kullanmasak da koyduğumuz boşluklar değişkenin içindedir yani a = “ asdfg. “ ile a = asdfg. Hiçbir farkı yok lakin rahat kullanım açısından bir değişkene

boşluk atayıp o şekilde kullanılabilir.

Tanımlanmamış değişken kullanırsak ne olur?

Böyle durumlarda hata vermek yerine boş bir string olarak algılanır.

```
all:
    echo $(ben_aslında_yoğum)
```

Makefile değişkenleri birleştirebilir miyim?

String append şeklinde tane += ile bunu sağlayabilirsiniz. Bir değişkene sonradan ekleme yapılabilir bu sayede.

```
aa := Because
aa += Trigger

all:
    echo $(aa)
```

Madem değişkenler değişebilir üzerine

yazılabilir mi?

Değikenlerin üzerine override isimli anahtar kelime ile yazılabilir. Sonrasında değişkeni değiştirmek için faydalı bir şey.

```
bir_numara = ananı zikim
#Üzerine yazılıyor
override bir_numara = did_override
option_two = not_override

all:
	echo $(option_one)
	echo $(option_two)
```

Tuhaf uzun değişkenler için bir atama değeri

define endef arasına istediğini yaz hepsi tek değişken.

Sadece bir kural ve ona bağlı olanlarda kullanılan bir değişken

Sadece all kuralı ve ona bağımlı kurallarda bir tanımlı.

```
all: bir = hazine

all:
	echo bir tanımlandı $(bir)
diğerleri:
	echo bir tanımlı değil $(bir)
```

Yukarıdaki özelliği kullanarak sadece belirli kurallarda tanımlı bir değişken ataması yapılabilir mi?

Evet % ile beraber kullanarak yapılması mümkün.

```
%.c: bir = Herkes ister ama .c ile biten kurallar alır

kaptım.c:
	echo bir tanımlandı $(bir)
diğerleri:
	echo bir tanımlı değil $(bir)
```

O kadar şey var if else yok mu?

Neden olmasın? Tabiki if not equal(ifneq) da var.

```
boşmu =
aptallar = $(boşmuş) #Sonunda boşluk var

all:
ifeq($(strip $(aptallar)),)
    echo “Soyunduktan sonra aptallar boş kaldı”
endif
ifeq($(boşmuş),)
    echo “boş olan şey boştur”
endif
```

Tanımlı mı diyebilir miyiz?

ifdef ve ifndef imdadımıza yetişiyor.

```
bar =
foo = $(bar)
```

```
all:
ifdef foo
    echo “Tanımlı”
endif
ifndef bar
    echo “Ama bar boş”
endif
```

Make Flagler test edebilir miyiz?

findstring ve MAKEFLAGS kullanarak bunu yapabiliriz.

```
bar =
foo = $(bar)

all:
ifneq (,$(findstring i , $(MAKEFLAGS)))
    echo “Bu testi de başardık”
endif
```

Hazır make fonksiyonları:

Fonksiyonlar genel olarak metin yönetimi

için var. Call fonksiyonu ile $(fn, değişkenler) veya ${fn, değişkenler} şeklinde kendi fonksiyonumuzu da yazabiliriz.

subst:

Stringteki belirli kelimeler yerine başka kelimeler yazmaya yarar

bar := ${subst a, Neden Olmasın, “ a bonbon a non a”}

Çıktı olarak Neden Olmasın bonbon Neden Olmasın non Neden Olmasın verir.

patsubst:

one := $(patsubst %.o,%.c,$(foo))

two := $(foo:%.o=%.c)

three := $(foo:.o=.c)

üçü de aynı şeyi yapar. Patsubst nin substden farkı wildcard kullanabiliyoruz.

foreach:

Anlatmaya gerek yok.

$(foreach var,list,text)

if:

```
    $(if this-is-not-empty,then!,else!)
```

call:

fonksiyonAdı = $(0) $(1) … şeklinde tanımlanır ve (call fonksiyonAdı, $(0), $(1) …) şeklinde kullanılır.

```
    sweet_new_fn = Değişken Adı: $(0)
Birinci: $(1) İkinci: $(2) Boş veri: $(3)
    all:
   @echo $(call sweet_new_fn, go, tigers)
```

shell:

Shell komutlarını çalıştırmaya yarar.

```
    all:
    @echo $(shell ls -la)
```

Kalan fonksiyonlar için link https://www.gnu.org/software/make/manual/html_node/Functions.html

Yaralı özellikler:

Include:

Başka make dosyalarını eklememize

yarar.

includes dosyaAdı…

şeklinde kullanılır.

vpath:

Dosya aramak için yol gösterir.

multiline:

Değişken tanımlarken sonuna / koyarak sıradaki satırın da dahil olmasını sağlayabiliriz.

.phony

Kural isimlerinin dosya olmasını durumunda o dosyaları görmezden gelir.