Contents

Türkmen Wikiye Hoşgeldiniz **1**
- Giriş: 1
- Logonun anlamı 1

Temel Kavramlar **3**
- Linux dağıtımı nedir 3
  - Sabit sürüm ve yuvarlanan sürüm 3
  - Uzun süreli destek (LTS) 3
- Linux dağıtımının temel yapısı 3
- Paket Sistemi ile İlişki 4
- Özgür Yazılım 4
  - Özgür Yazılımın Tanımı 4
  - Özgür Yazılımın Önemi ve Faydaları 4
  - Sonuç 5
- Özgür Yazılım ve Açık Kaynak Yazılımın Farkları 5
  - Özgür Yazılım 5
  - Açık Kaynak Yazılım 5
  - Farklar 6
- Temel Kavramlar sözlüğü 6

Ymp Paket Sistemi **9**
- Ymp paket sisteminin kurulumu 9
  - Kaynak kodun indirilmesi 9
  - Gerekli paketlerin kurulması 9
  - Yapılandırma 9
  - Derleme ve sisteme kurulma işlemi 9
- Ymp komut satırı kullanımı 9
  - Yardım çıktısı alma 10
  - Ymp değişkenleri 10
  - Ymp kabuğu 11
    - Açıklama satırları 11
    - Değişken tanımlama 11
    - Koşul tanımlama 11
    - Etiket tanımlama 11
- Ymp paket yapımı 12
  - Paket formatı 12
  - Derleme işlemi ve sandbox 13
  - ympbuild 13
    - Değişkenler 13
    - Diziler 13
    - İşlevler 13
  - Derleme dizinleri 14
  - Use flag kavramı 14

Bağımlılıklar 14
İşlevler 14
Özellik açma 15
Git tabanlı paketler 15
Depo oluşturma 15
Gpg anahtarı oluşturma 15
Gpg anahtarının repicent değerini bulma 16
Depo için dizin oluşturma 16
Depoyu yayınlama 16
Depoyu ekleme 16
Sysconf Dosyaları 17
Sistem 19
Servis Yönetimi 19
OpenRC 19
Kurulum 19
Basit kullanım 19
Docker içinde servis çalıştırma 20
Servis dosyası 20
İnit 20
Modüllerin yüklenmesi 21
Dosya sisteminin bağlanması 21
Servislerin başlatılması 21
Getty başlatılması 21
Dikkat edilmesi gerekenler 21
Ağ ve İnternet 22
udhcpc 22
OpenRC Servisi 22
NetworkManager 23
Kurulumu 23
Openrc servisi 23
Servisi elle başlatma 23
Wifi 23
Wpa-supplicant kurulumu 23
Bağlantının kurulması 24
Network Manager ile kullanılması 24
Bluetooth 25
Kurulum 25
Bağlantının kurulması 25
Otomatik bağlanma 26
Aygıtı kaldırma 26
Ssl sertifikası yükleme 26
Firefox için yükleme 26
Oturum ve Başlangıç 27

Kullanıcılar 27
Kullanıcı yönetimi 27
Kullanıcı ekleme ve silme 27
Kullanıcı ekleme 27
Sysconf yardımı ile kullanıcı ekleme 28
Kullanıcı silme 28
Parola ayarlama 28
Kullanıcılar arası geçiş 28
Root yetkisi 28
Root yetkisinin alınması 29
suid kavramı 29
setuid sistem çağrısı 29
suid engelleme 29
busybox su 30
Kullanıcı Kabuğu 30
Kabuğu değiştirme 30
Unix Kabuğu 30
Oturumlar 31
Elogind 31
Kurulumu 31
Oturumların listelenmesi 32
Kullanım 32
Oturum kontrolü 32
Yetkilendirme 33
Polkit 33
Kurulum 33
Polkit-agent 33
Kurallar 34
Eylemler 34
Polkit kullanımı 34
PAM 35
Yapılandırma: 35
Pam hata ayıklama 36
Pam modülü yazmak 36
Pam kullanımı 37
Önyükleyici 38
Grub 38
Kurulum 38
Yapılandırma 38
Grub konsolu 39
Diskler ve Dosya sistemleri 39
Dosya Sistemleri 39
ext4 39

Diski bağlama 39
Ext4 biçimlendirme 40
Günlüklemeyi kapatma 40
Btrfs 40
Disk bağlama 40
Btrfs biçimlendirme 40
Rootfs olarak kullanma 41
Subvolume kavramı 41
Bağlamak 41
Oluşturma 41
Listeleme 41
Silme 41
Yeniden adlandırma 42
Yedekleme (snapshot) 42
Zram 42
Yapılandırma 42
Kullanım 42
Zram Durumunu Kontrol Etme 43
Cryptsetup 43
Kurulum 43
Kullanımı 43
Parola değiştirme 43
Swap (Takas Alanı) 44
Swap alanı oluşturulması 44
Disk bölümünden swap oluşturulması 44
Dosyadan swap oluşturulması 44
Swap alanının etkinleştirilmesi 44
Swap alanının kapatılması 45
Network Block Device (nbd) 45
Sunucu tarafı 45
Disk İmajı Oluşturma 45
NBD Sunucusunu Başlatma 45
Kullanıcı tarafı 45
NBD Çekirdek Modülünü Yükleme 45
Bağlantıyı Ayarlama 45
Bölüm Listesini Güncelleme 46
Fuse 46
Kurulumu 46
Yapılandırma 46
Fuse ile dosya sistemi yazma 47
Disk Yönetimi 49
Udisks 49
Kurulum 49

Servisin başlatılması 49
Kullanım 49
Yapılandırma 50
Yetlkilendirme 50
Kurulum 50
Basit Kurulum 50
Uefi - Legacy tespiti 50
Disk Bölümlendirme 50
Dosya sistemini kopyalama 51
Bootloader kurulumu 51
Grub yapılandırması 52
Fstab dosyası 52
Yapılandırma 52
Dil ayarlama 52
Hostname ayarlama 53
Zaman dilimi ayarlama 54
Araçlar 54
syslog 54
Servisin başlatılması 54
Log yazma 55
ntpd 55
Servisin başlatılması 55
Yapılandırma dosyası 56
USB/IP 56
Kurulum 56
Sunucu tarafı 56
Kullanıcı tarafı 57
Container 57
Docker 57
Docker Kurulumu 57
Servisin başlatılması 58
Cgroup controller değiştirme 58
Basit kullanım 58
Çalışan containerleri görmek 59
Açık olan containeri kapatmak ve silmek 60
Durdurulmuş bir containeri başlatmak ve içine girmek 60
Cgroup 61
Cgroup oluşturma 61
Süreç ekleme 61
Cgroup silme 62
netns (Network Namespace) 62
netns oluşturma ve kaldırma 62
netns içerisinde komut çalıştırma 62

netns içine ağ arayüzü eklenmesi 62
netns arası iletişim sağlanması 63
Dış ağa bağlantı sağlanması 63
Geliştirme ortamı 63
Python 63
Python kurulumu 63
Pip etkinleştirilmesi 64
Venv ortamı 64
Vala 64
Valac kurulumu 64
Nano renklendirme desteği 65
C 65
Derleyici kurulumu 65
Gcc kurulumu 65
Clang kurulumu 65
Standart C Kütüphanesi (libc) 65
Glibc 66
Musl 66
Derleyici ayarlama 66
Rust 66
Kurulum 66
Proje oluşturma ve derleme 67
Javascript 67
npm kurulumu 67
Proje oluşturma 67
Bun kurulumu 68
Proje oluşturma 68
Donanım **69**
Sürücüler 69
Linux Firmware 69
linux-firmware kurulumu 69
DKMS 69
Kullanımı 69
dkms paketi hazırlama 70
Ekran Kartı 71
3D controller kapatmak 71
Nvidia 72
Nvidia sürücünün yüklenmesi 72
Çoklu ekran kartı kullanımı 75
Çoklu ekran kartı 75
Çoklu Ortam 75
Pipewire 75
Kurulumu 75

Uzak makinaya bağlanma 76
Ses seviyesi ayarı 76
Güç Kontrolü 78
Uyku modu 78
Uyku modunu tetikleme 78
Uyku moduna geçmeyi önleme 78
Laptop kapağı kapatma eylemi 78
Güç tasarrufu 79
İşlemci Governor Ayarları 79
İşlemci çekirdeklerini uykuya almak 79
Platform profile ayarı 79
Turbo boost ayarı 80
ACPI 80
Acpi tabloları 80
Acpi wakeup ayarları 80
Ekran parlaklığı kontrolü 81
vendor 81
video 81
native 81
Ekran parlaklığı 81
Grafik Arabirim **83**
Uygulamalar 83
AppImage 83
AppImage çalıştırmak 83
AppImage dosyalarını uygulama menüsüne eklemek 83
Flatpak 83
Kurulum 84
Depo ekleme 84
Uygulama yükleme 84
Uygulamaları güncelleme 84
Bubblewrap 84
Kullanımı 85
Başka dağıtım ile kullanma 85
Tek kullanımlık ev dizini 86
Araç Setleri 86
Gtk 86
Tema yükleme 87
Tema ayarlama 87
Tema test etme 87
Gdk backend ayarı 88
Overlay Scrollbar kapatma 88
Dokunma test modunu açma 88
Hata ayıklama 89

- Qt 89
  - Uygulama derleme 89
  - QT_QPA_PLATFORM 90
  - Temalama 90
- Masaüstü Ortamları 90
  - LXDE 90
    - Kurulumu 91
      - xinitrc ayarları 91
  - Xfce 91
    - Kurulumu 92
      - xinitrc ayarları 92
  - Cinnamon 92
    - Kurulumu 93
      - xinitrc ayarları 93
  - Kde 93
    - Kurulumu 94
      - xinitrc ayarları 94
- Pencere Yöneticileri 94
  - Weston 94
    - Yükleme 95
    - Çalıştırma 95
      - Weston'un Parametreleri 95
      - Yapılandırma 96
  - sway 96
    - Yükleme 96
    - Çalıştırma 97
    - Kullanımı 97
    - Ekran görüntüsü alma 97
    - Yapılandırma 97
  - Hyprland 97
    - Yükleme 98
    - Çalıştırma 98
    - Kullanımı 98
    - Ekran görüntüsü alma 99
    - Yapılandırma 99
- X11 99
  - Elle Başlatma 99
  - xinit ile başlatma 99
  - Uzak makinaya pencere yollamak 99
  - Xephyr 100
  - Xvfb 100
- HDPI 100
  - Dpi hesaplama 101

Gtk için 101
Qt için 101
Xft.dpi ayarı 101
Diğer Konular 103
Busybox ile Minimal Dağıtım Oluşturma 103
LD_PRELOAD 104
Musl libc 105
Kurulum 105
Kullanımı 105
Çalıştırılması 106
Türkmen Linux Hardened özellikleri 106
Komut listesi gizleme: 106
Canlı sistem: 106
Çekirdek: 106
Initramfs: 106
Sistem seviyesi önlemler: 107
Diğer 107
Sık karşılaşılabilen problemler 107
docbook-xsl / docbook-xml hataları 107
Segmentation fault hatası 107
aclocal-1.x bulunamadı hatası 108
Programlama 109
Bash dersi 109
Açıklama satırı ve dosya başlangıcı 109
Ekrana yazı yazalım 109
Parametreler 110
Değişkenler ve Sabitler 110
Diziler 112
Klavyeden değer alma 113
Koşullar 113
case yapısı 114
Döngüler 115
Fonksiyonlar 116
Dosya işlemleri 117
Boru hattı 119
Kod bloğu 119
select komutu 120
Birden çok dosya ile çalışmak 120
exec komutu 121
fd kavramı 121
Hata ayıklama 121
C Dersi 122
Derleme işlemi 122

- Açıklama satırı 122
- Girintileme 122
- İlk program 123
- Ekrana yazı yazma 123
- Değişkenler 123
- Diziler 124
- Klavyeden değer alma 124
- Koşullar 125
- Switch - Case 126
- Döngüler 126
- goto 127
- Fonksiyonlar 127
- Pointer ve Address kavramı 128
- Dinamik bellek yönetimi 130
- Struct 131
- Kütüphane dosyası oluşturma 132
- C++ kodunu C ile kullanma 134

Docker kullanımı 135

- Docker kurulumu 135
- Docker çalışma mantığı 136
- İmajlar 136
- Containerlar 137
- Uzak sunucuda çalışmak 139
- Volume kavramı 139
- Dockerfile 140
- Dockerfile yapısı 141

Git kullanımı 141

- Git kurulumu 141
- Git ne işe yarar 142
- Git kullanarak kaynak kodun indirilmesi 142
- Eski değişiklikleri görmek ve eski sürüme dönmek 142
- Sunucudaki güncel değişiklikleri almak 142
- Yeni commit oluşturma 143
- Git sunucusuna gönderme 143
- Branch kavramı 145
- Remote kavramı 146
- Squash commit kavramı 147

makefile dersi 147

- Genel bakış 147
- Değişken işlemleri 148
- Bölümler 149
- wildcard ve shell 150
- Birden çok dosya ile çalışma 150

Koşullar 151
Komut özellik ifadeleri 151
while ve for kullanımı 151
SHELL değişkeni 151
Python dersi 152
Açıklama satırları 152
Temel bilgiler 152
Yazı yazdırma 152
Değişkenler 153
String 153
Integer 154
Float 154
Boolean 155
Klavyeden değer alma 156
Koşullar 156
Diziler 157
While döngüsü 158
For döngüsü 159
İşlevler 160
Sınıflar 161
Dosya işlemleri 162
Modüller 162
Vala dersi 164
Girintileme 165
Açıklama satırı 165
Ekrana yazı yazdırma 165
Değişken türleri 166
Diziler 166
Klavyeden değer alma 168
Koşullar 168
Döngüler 169
Fonksiyonlar ve parametreler 170
Sınıf kavramı 172
Super sınıf 172
Signal kavramı 173
Namespace kavramı 174
Kütüphane oluşturma 175
Gobject oluşturma 176

Türkmen Wikiye Hoşgeldiniz

Bu wikiye katkı sağlamak için https://codeberg.org/turkmen-linux/wiki/ adresine pull request gönderebilirsiniz.

Bu dokümana https://turkmen-linux.github.io/wiki/rst2pdf.pdf adresinden çevirimdışı pdf formatında da erişebilirsiniz.

Bu doküman python-sphinx. kullanılarak oluşturulmuştur. Dokümanın kendisi GPL (V3) lisansına sahiptir.

Giriş:

Türkmen Linux bağımsız tabanlı bir GNU/Linux dağıtımıdır. Kendisine ait YMP (Yerli ve Milli Paket) sistemini kullanır. Türkmen Linux Türkiye originli bir dağıtımdır. □□ Geliştirilme sebebi yeni nesil teknolojileri kullanan bağımsız bir dağıtım oluşturmak ve kullanım kolaylığı sağlamaktır.

- Bütün paketler header dosyaları ve static kütüphaneler ile beraber gelmektedir. Bu sayede derleme yapması diğer dağıtımlardan daha kolaydır.
- Hızlı çalışan karışık (ikili ve kaynak) paket sistemine sahiptir. İsteyenler kaynak koddan paket kurarken isteyenler derlenmiş paketleri kullanabilirler.
- Varsayılan olarak **sıkılaştırılmış** (hardened) linux dağıtımıdır. Bu sayede ek çaba sarf etmeden güvenle kullanılabilir.

Logonun anlamı

Türkmen linuxun logosu mavi zemin üzerine 3 hilalden oluşmaktadır. Logodaki 3 hilal osmanlı döneminde (1517-1793 arasında) kullanılan bayraktan esinlenilmiştir. Mavi renk (#1aa3ff) ise türkmen mavisidir.

Kaynak kod yansılarımız:

- https://github.com/turkmen-linux
- https://codeberg.org/turkmen-linux

Temel Kavramlar

Linux dağıtımı nedir

Linux dağıtımı (GNU/Linux) linux çekirdeğini kullanan ve GNU araç setini barındıran bir işletim sistemidir. Buna dağıtım denmesinin sebebi özgür yazılım olması ve isteyenlerin istediği gibi düzenleme ve dağıtma yetkisinin bulunmasıdır.

Taban kavramı ise bir gnu/linux dağıtımının kullandığı temel uygulama ve ayarların standartlarıdır. Örneğin debian tabanı paket sistemi olarak apt kullanır. glibc kütüphanesini kullanır ve sabit sürümlüdür (fixed release) ve uzun süreli destek (LTS) yapısına göre geliştirilir. Alpine tabanı ise glibc yerine musl kullanır. Yuvarlanan sürümlüdür (rolling release).

Dağıtımların kendine ait tabanı bulunabilir veya başka bir dağıtımın tabanını kullanabilir.

Sabit sürüm ve yuvarlanan sürüm

Sabit sürümlü dağıtımlarda paketler belli aralıklarla toplu olarak güncelleme alırlar. Genellikle daha kararlıdırlar fakat daha yavaş güncelleme alırlar. Sabit sürümlü dağıtımlara örnek olarak **debian** ve türevleri örnek gösterilebilir.

Yuvarlanan sürümlü dağıtımlar ise sürekli olarak güncelleme alırlar ve herhangi bir sürüm numarasına sahip değillerdir. Bu daha az kararlı olmasına sebep olabilirken daha güncel bir sistem sağlayabilir. Yuvarlanan sürümlü dağıtımlara örnek olarak **archlinux** gösterilebilir.

Bağımsız tabanlı dağıtım tasarlamak için yuvarlanan sürümlü dağıtım tasarlamak daha basittir ve daha az uğraş gerektirir.

Uzun süreli destek (LTS)

Bazı gnu/linux dağıtımları üzün süreli desteğe sahiptirler. Uzun süreli desteğe sahip sistemler daha kararlıdır fakat genellikle eski paketler içerirler. Genellikle sık güncelleme yapılması istenmeyen alanlarda veya kararlılığın önemli olduğu alanlarda tercih edilir. Yuvarlanan sürümlü dağıtımların uzun süreli desteği bulunmaz. Uzun süreli destek özelliğine sahip olan uygulama ve sistemlerde bulunan güvenlik açıkları ve hatalar güncel sürümden uyarlama yolu ile eklenir. (backports) Bu uyarlamalar sayesinde açıklar ve eksikler olmadan sistem sorunsuz bir şekilde çalıştırılabilir.

Yazar notu: Bana sorarsanız LTS kullanmayın :D

Linux dağıtımının temel yapısı

Bir linux dağıtımını oluşturan tüm parçalar paketlerden oluşur. Bu bölüm linux dağıtımının temel yapısı ve paket sistemi ile ilişkisi anlatılacaktır.

Sıradan bir linux dağıtımını 4 temel parçada ele alabiliriz.

Kernel, bir Linux dağıtımının en temel bileşenidir. Donanım kaynaklarını yönetir, sürücüleri yükler, sistem çağrılarını işler ve diğer yazılım bileşenleriyle etkileşim kurar. Linux kerneli, genellikle dağıtım geliştiricileri tarafından özelleştirilir ve optimize edilir. Yeni sürümler, performans iyileştirmeleri, güvenlik yamaları ve donanım uyumluluğu gibi çeşitli iyileştirmeler içerebilir.

Initramfs (initial RAM filesystem), sistem başlatma işlemi sırasında kullanılan geçici bir dosya sistemidir. Initramfs, ana dosya sistemine erişim sağlamadan önce gerekli sürücüleri yükler ve sistem başlatma sürecini tamamlar. Bu, özellikle karmaşık donanım yapılandırmalarına sahip sistemlerde, önyükleme sırasında gerekli olan sürücülerin doğru şekilde yüklenmesini sağlar.

Servis yöneticisi, sistemde çalışan hizmetlerin yönetiminden sorumludur. Bu hizmetler genellikle sistem başlatıldığında otomatik olarak başlatılır ve gerektiğinde durdurulur. Servis

yöneticisi ayrıca hizmetlerin bağımlılıklarını yönetir, bu da sistem performansını ve güvenilirliğini artırır. Linux dağıtımlarında yaygın olarak kullanılan servis yöneticileri arasında systemd, OpenRC, SysVinit ve Upstart bulunmaktadır.

Masaüstü ortamı, kullanıcı arabirimini sağlar ve kullanıcıların sistemle etkileşimde bulunmasını sağlar. Bu, pencere yönetimi, dosya yönetimi, uygulama başlatma ve sistem ayarlarını yapılandırma gibi görevleri içerir. Linux dünyasında birçok farklı masaüstü ortamı bulunmaktadır, bunlar arasında GNOME, KDE Plasma, Xfce, LXQt, Cinnamon, MATE ve daha fazlası yer almaktadır.

Paket Sistemi ile İlişki

Linux dağıtımlarındaki temel parçaların çoğu, paketler şeklinde dağıtılır ve yönetilir. Paketler, yazılımın kurulumunu, güncellemesini ve kaldırılmasını kolaylaştıran birimlerdir. Bir Linux dağıtımı, genellikle bir paket yöneticisi aracılığıyla paketleri arar, yükler, günceller ve kaldırır. Paket yöneticisi, paket bağımlılıklarını yönetir ve çatallanma, güvenlik güncellemeleri ve yazılım dağıtımı gibi işlemleri kolaylaştırır.

Örneğin, Debian tabanlı dağıtımlarda (örneğin, Ubuntu), apt (Advanced Package Tool) paket yöneticisi kullanılırken, Red Hat tabanlı dağıtımlarda (örneğin, Fedora), yum (Yellowdog Updater Modified) veya daha yeni dağıtımlarda dnf (Dandified Yum) kullanılabilir.

Özgür Yazılım

Özgür yazılım, modern bilgi çağında bireylerin ve toplulukların bilgiye erişimini, kullanımını ve paylaşımını teşvik eden bir yazılım yaklaşımıdır. Bu yaklaşım, yazılım kullanıcılarının özgürlüklerini korurken aynı zamanda yazılımın kalitesini ve güvenliğini artırmayı amaçlar. Özgür yazılım felsefesi, Özgür Yazılım Vakfı (FSF) ve benzeri kuruluşlar tarafından desteklenmektedir.

Özgür Yazılımın Tanımı

Özgür yazılım, kullanıcıların yazılımı çalıştırma, inceleme, paylaşma ve değiştirme özgürlüğüne sahip oldukları bir yazılım türüdür. Bu özgürlükler, yazılımın kaynak kodunun açık olması sayesinde mümkün hale gelir. Kullanıcılar, yazılımı istedikleri gibi kullanabilir, ihtiyaçlarına göre özelleştirebilir ve hatta geliştirebilirler. Özgür yazılımın temel özellikleri şunlardır:

1. **Özgürlük**: Kullanıcıların yazılımı istedikleri amaçlar için kullanma özgürlüğü vardır. Bu özgürlük, yazılımın kontrolünü kullanıcıya verir ve kısıtlamalardan kurtulmayı sağlar.
2. **Açık Kaynak Kodu**: Özgür yazılımın kaynak kodu herkes tarafından erişilebilir ve inceleme yapılabilir durumdadır. Bu sayede, yazılımın güvenliği ve kalitesi topluluk denetimi altında artar.
3. **Topluluk Katkısı**: Özgür yazılım genellikle büyük ve aktif bir topluluk tarafından desteklenir. Kullanıcılar, projeye katkıda bulunabilir, hataları bildirebilir ve yazılımın gelişimine katkı sağlayabilirler.
4. **Lisanslama**: Özgür yazılım genellikle Özgür Yazılım Vakfı (FSF) tarafından belirlenen özgür yazılım lisansları altında dağıtılır. Bu lisanslar, kullanıcıların özgürlüklerini koruyan belirli koşullar içerir.

Özgür Yazılımın Önemi ve Faydaları

Özgür yazılımın önemi, sadece yazılım kullanıcılarının özgürlüklerini korumakla sınırlı değildir. Ayrıca, şu faydaları da sağlar:

1. **Demokratik Katılım**: Özgür yazılım, yazılım geliştirme sürecine demokratik bir katılım sağlar. Herkes, yazılımı inceleyebilir, hataları rapor edebilir ve geliştirmeye katkıda bulunabilir. Bu, yazılımın toplumun ihtiyaçlarına daha iyi yanıt vermesini sağlar.

2. **İnovasyon ve İşbirliği**: Açık kaynak kodu, farklı geliştiricilerin bir araya gelerek işbirliği yapmasını ve yenilikçi çözümler üretmesini teşvik eder. Yazılımın kaynak koduna erişebilme özgürlüğü, yeni fikirlerin hızla yayılmasını sağlar.
3. **Ekonomik Faydalar**: Özgür yazılım, kuruluşların yazılım maliyetlerini azaltmalarına ve buna bağlı olarak daha rekabetçi olmalarına olanak tanır. Ayrıca, yazılımın esnekliği ve özelleştirilebilirliği, kuruluşların ihtiyaçlarına daha iyi uyacak çözümler bulmalarını sağlar.
4. **Erişilebilirlik**: Açık kaynak kodu, yazılımın herkes tarafından erişilebilir ve kullanılabilir olmasını sağlar. Bu, özellikle gelişmekte olan ülkelerde ve eğitim kurumlarında yazılıma erişimi artırabilir.

Sonuç

Özgür yazılım, kullanıcıların özgürlüklerini koruyan, toplumun ortak çıkarlarına hizmet eden ve bilgiye erişimi teşvik eden bir yazılım felsefesidir. Bu yaklaşım, modern bilgi çağında bilgiye erişimin ve paylaşımının önemini vurgular. Özgür yazılımın benimsenmesi, daha adil, güvenli ve yenilikçi bir dijital geleceğin temellerini atabilir.

Özgür Yazılım ve Açık Kaynak Yazılımın Farkları

Özgür yazılım ve açık kaynak yazılım terimleri genellikle birbirinin yerine kullanılsa da, aslında farklı kavramları ifade ederler. İkisi de yazılımın erişilebilirliği ve kullanımı konusunda benzer özelliklere sahip olabilir, ancak felsefi ve lisanslama açısından farklıdırlar.

Özgür Yazılım

Özgür yazılım, kullanıcıların yazılımı çalıştırma, inceleme, paylaşma ve değiştirme özgürlüğüne sahip oldukları bir yazılım türüdür. Özgür yazılım felsefesi, kullanıcıların özgürlüklerini korumayı ve yazılımın toplumun ortak çıkarlarına hizmet etmesini amaçlar. Özgür yazılımın temel özellikleri şunlardır:

1. **Özgürlük**: Kullanıcıların yazılımı istedikleri gibi kullanma özgürlüğü vardır. Bu, yazılımın kaynak kodunun açık olması ve kullanıcıların yazılımı istedikleri gibi değiştirebilme yeteneğiyle sağlanır.
2. **Topluluk Katkısı**: Özgür yazılım projeleri genellikle büyük ve aktif bir topluluk tarafından desteklenir. Kullanıcılar, projeye katkıda bulunabilir, hataları bildirebilir ve yazılımın gelişimine katkı sağlayabilirler.
3. **Lisanslama**: Özgür yazılım genellikle Özgür Yazılım Vakfı (FSF) tarafından belirlenen özgür yazılım lisansları altında dağıtılır. Bu lisanslar, kullanıcıların özgürlüklerini koruyan belirli koşullar içerir.

Açık Kaynak Yazılım

Açık kaynak yazılım, yazılımın kaynak kodunun herkes tarafından erişilebilir ve inceleme yapılabilir olduğu bir yazılım türüdür. Açık kaynak yazılım felsefesi, yazılımın kalitesini artırmak ve geliştirme sürecini şeffaf hale getirmeyi amaçlar. Açık kaynak yazılımın temel özellikleri şunlardır:

1. **Açık Kaynak Kodu**: Yazılımın kaynak kodu herkes tarafından erişilebilir ve inceleme yapılabilir durumdadır. Bu, yazılımın güvenliği ve kalitesinin topluluk denetimi altında artmasını sağlar.
2. **Topluluk İşbirliği**: Açık kaynak yazılım projeleri genellikle geliştiricilerin bir araya gelerek işbirliği yaptığı büyük bir topluluk tarafından desteklenir. Bu, yazılımın hızla gelişmesini ve iyileştirilmesini sağlar.
3. **Esneklik ve Özelleştirme**: Açık kaynak yazılım, kullanıcıların yazılımı istedikleri gibi özelleştirmelerine ve ihtiyaçlarına göre uyarlamalarına olanak tanır.

Farklar

Özgür yazılım ve açık kaynak yazılım arasındaki temel fark, felsefi ve lisanslama yaklaşımlarıdır. Özgür yazılım, kullanıcıların özgürlüklerini vurgularken, açık kaynak yazılım daha çok yazılımın geliştirilmesi ve kalitesinin artırılmasına odaklanır. Özgür yazılım genellikle FSF tarafından belirlenen özgür yazılım lisansları altında dağıtılırken, açık kaynak yazılım genellikle daha çeşitli lisanslar altında yayımlanabilir.

Temel Kavramlar sözlüğü

Linux'a başlarken sıkça karşılaşabileceğiniz temel kavramlar:

- **ACPI** (Advanced Configuration and Power Interface), bilgisayar donanımının yönetimini sağlayan bir endüstri standardıdır ve enerji tüketimini optimize etmek için kullanılır.
- **Ağ Aygıtları ve Arayüzler** (Network Devices and Interfaces): Ethernet kartları, kablosuz adaptörler gibi ağ bağdaştırıcıları ve bunların bağlandığı ağ arayüzleri.
- **APM** (Advanced Power Management), eski bilgisayar sistemlerinde enerji yönetimini sağlayan bir standarttır, ancak ACPI gibi daha modern yönetim teknolojileriyle yer değiştirmiştir.
- **Bağımlılıklar** (Dependencies): Yazılım paketlerinin diğer paketlere veya kütüphanelere bağımlılıkları. Paket yöneticileri bu bağımlılıkları otomatik olarak yönetir.
- **Bellek Yönetimi** (Memory Management): Sistem belleğinin kullanımını izleme, optimize etme ve yönetme işlemi. Bellek sızıntıları ve düşük bellek uyarıları gibi sorunları tespit etmek için önemlidir.
- **BIOS** (Basic Input/Output System): Bir bilgisayarın donanımını başlatmak ve temel giriş/çıkış işlevlerini sağlamak için kullanılan bir yazılımdır.
- **Çekirdek Derleme** (Kernel Compilation): Linux çekirdeğini belirli bir sistem için özelleştirmek veya yeni özellikler eklemek için çekirdeği yeniden derleme süreci.
- **Çekirdek** (Kernel): Linux'un merkezi bileşeni. Donanım ile iletişim kurar ve diğer yazılımların çalışmasını sağlar.
- **Daemon**: Arka planda çalışan süreçler. Örneğin, web sunucuları (Apache, Nginx) ve veritabanı sunucuları (MySQL, PostgreSQL) gibi.
- **Dağıtım** (Distribution): Linux'un bir versiyonu veya paketi. Örneğin, Ubuntu, Fedora, Debian gibi.
- **Dosya Sistemleri** (File Systems): Linux'un farklı depolama cihazları üzerinde verileri organize etmek için kullandığı yapılar. Örneğin, ext4, XFS, Btrfs gibi.
- **Dosya ve Dizinler**: Linux'ta her şey dosya veya dizin olarak temsil edilir. /home, /etc, /var gibi dizinler ve dosya uzantıları.
- **Dosya İzinleri** (File Permissions): Dosyaların ve dizinlerin erişim izinleri. Kullanıcılar ve gruplar için okuma, yazma ve çalıştırma izinleri belirlenebilir.
- **GUI** (Graphical User Interface): Grafiksel kullanıcı arayüzü. GNOME, KDE gibi masaüstü ortamları.
- **Güvenlik Duvarı** (Firewall): Ağ trafiğini kontrol eden güvenlik duvarı yazılımı. IPTables, UFW gibi araçlar kullanılabilir.
- **Hizmetler** (Services): Sistemde çalışan ve belirli bir görevi yerine getiren yazılım parçaları. Önyükleme sırasında otomatik olarak başlayabilirler.
- **Kernel Modülü** (Kernel Module): Linux çekirdeğine dinamik olarak eklenen sürücüler veya özellikler. Örneğin, bir donanım sürücüsü bir kernel modülü olarak yüklenir.

- **Kernel Panic**: Linux Kernel'inin kritik bir hatayla karşılaştığında kendini korumak için durması. Genellikle sistemde ciddi bir sorun olduğunu gösterir.
- **Kernel Parametreleri** (Kernel Parameters): Linux çekirdeğine başlatma sırasında geçirilen özel parametreler. Performansı, güvenliği veya donanım uyumluluğunu etkileyebilirler.
- **Komut Satırı Geçmişi** (Command Line History): Kullanıcıların daha önce çalıştırdıkları komutların bir listesi. Geçmiş komutlarınızı tekrar kullanmanızı sağlar.
- **Konsol** (Console): Terminal penceresi veya gerçek konsol donanımı. Komutları yazıp çalıştırmanızı sağlar.
- **Kök Dosya Sistemi** (Root Filesystem): Linux'un en üst düzey dosya sistemi. "/" olarak adlandırılır ve tüm diğer dosya ve dizinler bu kök dizinin altında yer alır.
- **Kullanıcı** (User): Sistemdeki normal kullanıcı hesapları.
- **Kullanıcı Oturumu** (User Session): Bir kullanıcının bir bilgisayarda oturum açtığı ve işlemlerini gerçekleştirdiği zaman aralığı.
- **Önyükleyici** (Bootloader): Bilgisayarın işletim sistemini başlatmak için kullanılan yazılım parçasıdır. Temel görevi, bilgisayar açıldığında BIOS veya UEFI tarafından yüklenen ve ardından işletim sistemi yüklenirken kullanılan bir programdır.
- **Paket Yöneticisi** (Package Manager): Yazılım paketlerini yöneten araç. Yeni yazılım yüklemek veya mevcut yazılımları güncellemek için kullanılır.
- **Root Kullanıcı** (Root User): Linux'un sistem yöneticisi. Tam yetkiye sahip kullanıcı hesabıdır.
- **SSH Anahtarları** (SSH Keys): SSH ile güvenli erişim için kullanılan kimlik doğrulama anahtarları.
- **SSH** (Secure Shell): Uzak sistemlere güvenli bir şekilde erişmek için kullanılan bir protokol.
- **Shell betikleri** (Shell Scripts): Kabuk komutlarını içeren ve belirli bir görevi otomatikleştirmek için kullanılan metin dosyaları.
- **Shell**: Komutları çalıştırmak için kullanılan program. Bash, Zsh gibi popüler kabuklar bulunur.
- **Sudo**: Belirli komutları yönetici (root) yetkileriyle çalıştırmak için kullanılan bir komut. Genellikle güvenlik nedenleriyle tavsiye edilir.
- **Takas Alanı** (Swap Space): Fiziksel bellek (RAM) yetersiz olduğunda kullanılan sanal bellek alanı. Disk üzerindeki özel bir bölüm veya dosya olabilir.
- **Terminal Komutları**: ls (list), cd (change directory), mkdir (make directory), rm (remove), cp (copy), mv (move) gibi komutlar.
- **Terminal**: Komut satırı arayüzü. Metin tabanlı komutlar kullanarak işlemleri gerçekleştirmenizi sağlar.
- **X11 Yöneticisi** (X11 Manager): GUI uygulamalarının ekrana çizilmesini ve kullanıcı etkileşimini yöneten bir sistem bileşeni.
- **Yapılandırma Dosyaları**: Sistem ayarlarını ve yapılandırmalarını içeren metin dosyaları. Örneğin, /etc/hosts, /etc/fstab gibi.
- **Yapılandırma Yöneticisi** (Configuration Manager): Sistem ayarlarını yöneten araçlar. Örneğin, NetworkManager ağ yapılandırmalarını yönetir.
- **Yazılım Depoları** (Software Repositories): Yazılım paketlerini depolayan ve dağıtan sunucular. Paket yöneticileri bu depoları kullanarak yazılım yükler.

Ymp Paket Sistemi

Ymp paket sisteminin kurulumu

Ymp paket sistemini kaynak koddan derlemek için şu adımlar izlenmelidir.

Kaynak kodun indirilmesi

Kaynak koda https://gitlab.com/turkmen-linux/devel/sources/ymp adresinden ulaşabilirsiniz.

Öncelikle kaynak kodu *git clone* komutu ile indirelim.

```
$ git clone https://gitlab.com/turkmen-linux/devel/sources/ymp.git
```

Gerekli paketlerin kurulması

Debian tabanlı dağıtımlar için aşağıdaki paketler kurulmalıdır. (testing/unstable için)

```
# Derleyiciler
$ apt install meson gcc valac --no-install-recommends -y
# Derleme bağımlılıkları
$ apt install libarchive-dev libreadline-dev libcurl4-openssl-dev \
    libbrotli-dev --no-install-recommends -y
```

Yapılandırma

Kaynak kod *meson* komutu ile yapılandırılır.

```
$ meson setup build --prefix=/usr
    -Dshared=true
# Burada -Dshared=true veya -Dstatic=true belirtilmelidir.
```

İstenilen özellikleri *-Dözellik=durum* şeklinde belirterek ayarlayabilirsiniz. Özelliklerin listesine kaynak kod içerisinde bulunan meson_options.txt dosyasından ulaşabilirsiniz.

Derleme ve sisteme kurulma işlemi

ninja komutu kullanılarak derleme işlemi yapılır. Derleme sonrasında *ninja install* komutu ile sisteme kurulur. Son olarak *ldconfig* komutu kullanılarak kütüphanesini sistem kütüphane önbelleğini güncellenir.

```
# Derleme işlemi
$ ninja -C build
# Kurulma işlemi
$ ninja install -C build
$ ldconfig
```

Ymp komut satırı kullanımı

Ymp terminal üzerinden komut kullanarak çalıştırarak kullanılır.

```
$ ymp <işlem adı> <parametreler>
```

Yardım çıktısı alma

Tüm ymp işlemlerinin listesine ulaşmak için **ymp help** komutu kullanılır.

```
$ ymp help
...
-> install : Install package from source or package file or repository
...
```

Örneğin ymp kullanarak *git* paketini yükleyelim. Bunun için *ymp install git* komutunu çalıştırmamız gerekmektedir. Komuta ait parametreleri listelemek için **--help** parametresi eklememiz gereklidir.

```
$ ymp install --help
-> Aliases:install / it / add
-> Usage: ymp install [OPTION]... [ARGS]...
-> Install package from source or package file or repository
-> Options:
->   --ignore-dependency : disable dependency check
->   --ignore-satisfied : ignore not satisfied packages
->   --sync-single : sync quarantine after every package installation
->   --reinstall : reinstall if already installed
->   --upgrade : upgrade all packages
->   --no-emerge : use binary packages
-> Common options:
->   -- : stop argument parser
->   --allow-oem : disable oem check
->   --quiet : disable output
->   --ignore-warning : disable warning messages
->   --debug : enable debug output
->   --verbose : show verbose output
->   --ask : enable questions
->   --no-color : disable color output
->   --no-sysconf : disable sysconf triggers
->   --sandbox : run ymp actions at sandbox environment
->   --help : write help messages
```

Ymp işlemlerinin kısa adları da bulunur. Bu sayede komutu daha kısa şekilde yazıp kullanmamız mümkündür.

```
# Bu ifade ile bir sonraki aynı anlama gelir.
$ ymp it git
$ ymp install git
```

Ymp değişkenleri

Ymp çalışırken kendi içerisinde çeşitli değişkenler tanımlar ve bunları kullanarak çalıştırılacak işlemin özelliklerini belirler. Örneğin Ymp paketleri kurarken derleme yapılması istenmiyorsa **no-emerge** değişkeni **true** olarak ayarlanmalıdır. Değişkenler 2 şekilde ayarlanabilir. İlk olarak komut çalıştırılırken parametre eklenebilir.

Not: Tanımlanmamış olan tüm değişkenler **false** olarak kabul edilir.

Diğer yol ise **/etc/ymp.yaml** dosyasında ymp bölümüne ekleyebilirsiniz.

Ymp kabuğu

Ymp kendi içerisinde komut satırına sahiptir. Bu sayede istenilen işi bir betik dosyasına yazıp çalıştırmanız mümkündür. Ymp kabuğu başlatmak için **ymp shell** komutu kullanılır.

```
$ ymp shell
-> Ymp >> install git
```

Ymp kabugu ymp komutlarını ve parametrelerini kullanarak çalışır. Kabuk üzerinde normal işlemlerin yanında fazladan sadece kabukta çalışan ek işlemler bulunur. Bunlarla ilgili bilgi almak için **ymp help --all** komutu çıktısından yararlanabilirsiniz.

Ymp kabuğu kendi içerisinde basit bir programlama dili yorumlayıcısı barındırır. Bunu kapsamlı bir programlama dili olarak düşünmemekte fayda vardır. Çünkü Bir programlama dilinin sahip olduğu özelliklerin çoğundan mahrumdur ve sadece ymp komutlarının çalıştırılması üzerine tasarlanmıştır.

Açıklama satırları

ile başlayan satırlar açıklama satırıdır. Bununla birlikte **:** komutu işlevsizdir ve açıklama satırı olarak kullanılabilir.

```
# Bu bir açıklama satırıdır.
: Bu da bir açıklama satırıdır.
```

Değişken tanımlama

Değişken tanımlamak için **set** kullanılır. **read** komutu ise klavyeden alınan değeri değişken olarak atar. Bir değişkenin değerini almak için **get** kullanılır. Değişkenler kullanılırken başlarına **$** işareti konulur.

Not: Değişkenlerin herhangi bir karakter kısıtlaması bulunmamaktadır. Sayı, karakter veya türkçe karakter içerebilirler. (Emoji bile kullanabilirsiniz :D)

Koşul tanımlama

if ifadesi ile başlayan satır koşul satırıdır ve **endif** ifadesi gelene kadarki satırlar koşul sağlandığında çalıştırılır.

```
read var
if eq 12 $var
    echo sayı 12ye eşit
endif
```

Etiket tanımlama

label ifadesi ile kabuk betiğinde etiket tanımlanabilir. Daha sonra **goto** ifadesi kullanılarak bu etikete gitmek mümkündür. ymp kabuğunda döngüler bu şekilde sağlanır. Aşağıda siz 0 yazana kadar yazdığınızı ekrana yazan ymp kabuğu betiği mevcuttur.

```
label test
read var
if eq $var 0
    exit
endif
echo $var
goto test
```

Ymp paket yapımı

Ymp paketleri 3 türe ayrılır.

- kaynak paketler
- ikili paketler
- derleme talimatları

Derleme talimatları bizim tarafımızdan yazılan **ympbuild** dosyalarıdır. Bu dosyalar sayesinde ymp hangi kaynak kodun kullanılacağı, sürüm numarası, adı gibi bilgileri anlayabilir.

ympbuild dosyaları aslında birer bash betiğidir. Bu sayede kolay yapılıdır ve bash programlama bilgisi paket yapımı için yeterli olur.

ympbuild dosyası oluşturmak için **ymp template** komutundan yararlanabilirsiniz. Bu komut size şablon olarak ympbuild dosyası üretir. Bu sayede sadece üzerinde düzenleme yaparak kolayca paket yapabilirsiniz.

Paket formatı

Derleme talimatlarından kaynak ve ikili paketler üretmek için **ymp build** komutu kullanılır. Bu işlem önce kaynak kodu indirir ve doğrular. ardındar istenilen komutlara göre derleme işlemi yapıp paket üretir.

Örnek **ympbuild** dosyası aşağıdaki gibidir

```
#!/usr/bin/env bash
name=example
version=1.0
release=1
url='https://example.org'
description='example package'
email='your-name@example.org'
maintainer='linuxuser'
depends=(foo bar)
source=("https://example.org/source.zip"
        "some-stuff.patch"
)
arch=(x86_64 aarch64)
sha256sums=('bb91a17fd6c9032c26d0b2b78b50aff5'
    'SKIP'
)
license=('GplV3')
prepare(){
    ...
}
setup(){
    ...
}
build(){
    ...
}
package(){
   ...
}
```

Burada paketin bilgileri ve nasıl derleneceğini tanımlayan işlevleri görebilirsiniz.

Derleme işlemi ve sandbox

Bir ympbuild dosyasını derlemek için aşağıdaki gibi bir komut kullanılmalıdır.

```
$ fdir=./repo/stuff-package/
$ ymp build "$fdir" --sandbox --shared="$fdir"
```

Burada ilk önce paketin derleneceği dizini değişkene atadık. Paket derlenirken **sandbox** özelliğini açmak için **--sandbox** parametresi ekledik ve sandbox içerisine paketin derleneceği dizini erişilebilir hale getirmek için **--shared** parametresine dizinin konumunu yerleştirdik.

Bu komut sandbox olmadan şu şekilde görünürdü.

```
$ ymp build ./repo/stuff-package/
```

Burada sandbox zorunlu değildir fakat paketlerin düzgünce derlenmesi için kullanmanızı şiddetle öneririm.

ympbuild

Değişkenler

- **name** : Paketin adını belirtir.
- **version** : Paket sürümünü belirtir.
- **release** : Paket numarasını belirtir. Ymp güncellik kontrolü için sadece bu değere bakar.
- **url** : Paketin anasayfasını belirtir. Bu değer kullanılmaz.
- **description** : Paket açıklamasını belirtir.
- **email** : Paketçinin email adresidir.
- **maintainer** : Paket bakımcısının adıdır. (veya nickname)

Not: ymp derleme işleminin ardından paket boyutunu azaltmak için **strip** işlemini otomatik olarak uygulamaktadır. Bu durum bazı paketlerin bozulmasına sebep olabilir. Bunu engellemek için **dontstrip** değişkeni tanımlayarak bir değer atayabilirsiniz.

Diziler

- **depends** : Paket bağımlılıklarını belirtir
- **source** : Paket kaynak kodları listesini belirtir
- **sha256sums** : Paket sha256sum değeri listesidir. **SKIP** olan elemanları görmezden gelinir.
- **uses** ve **uses_extra** : use flag listesidir.
- **arch** : Desteklenen mimari listesidir.

İşlevler

- **prepare** : Hazırlık aşamasıdır. Burada kaynak kod yamaları uygulanır.
- **setup** : Kaynak kod yapılandırma aşamasıdır.
- **build** : Kodun derlendiği aşama burasıdır.
- **package** : Kaynak kodun paketleme dizinine kurulduğu aşamadır.

Derleme dizinleri

Her derlemenin **/tmp/ymp-build/<build-id>** içinde kendi derleme dizini vardır. build-id aslında ympbuild dosyasının md5sum'udur, bu nedenle ympbuild'i değiştirirseniz build-id değişir. Derleme dizini **HOME** çevresel değişkeni oarak tanımlanır. Bu sayede sadece **cd** komutunu kullanarak derleme dizinine geri dönebilirsiniz.

Derlenen kaynak kodlar paketlenirken **/tmp/ymp-build/<build-id>/output** dizinine kurulmalıdır. Bu dizin **installdir** ve **DESTDIR** çevresel değişkeni ile tanımlanır. Bu sayede **make install** gibi komutlara herhangi bir ek parametre vermenize gerek kalmaz.

Not: /tmp dizini genellikle ramdisk olarak bağlı olduğu için derleme sırasında ram dolabilir. Bunu engellemek için aşağıdaki gibi bir komut kullanabilirsiniz.

```
$ rm -rf /tmp/ymp-build
$ mkdir /home/linuxuser/ymp-build
# Bunu sistemi her başlattığınızda tekrarlamanız gerekebilir.
$ ln -s /home/linuxuser/ymp-build /tmp/ymp-build
```

Use flag kavramı

Paketlerde özellik tanımları yapmak için **uses** ve **uses_extra** dizileri tanımlayabilirsiniz. Bu özellikler isteğe bağlı açılıp kapatılabilirler. Bu sayede isteyenler paketleri istedikleri özelliklerle kullanabilirler.

```
...
uses=(foo bar)
uses_extra=(bazz)
foo_depends=(foo bazz)
...
setup(){
    ../configure --prefix=/usr \
    $(use_opt foo --with-foo --without-foo)
}
...
package(){
    ...
    if use bar ; then
        install stuff ${DESTDIR}/bin/stuff
    fi
}
```

Bağımlılıklar

Tanımlanan her özellik için **xxx_depends** şeklinde dizi tanımlayarak o özelliğin ek bağımlılıkları belirtilebilir. Bu sayede özelliği açtığımızda hangi ek paketlere ihtiyaç duyduğumuzu anlamamız mümkün olur.

İşlevler

Burada **use_opt** özelliğin açık olup olmama durumuna göre çalışır. Kullanımı şu şekildedir:

```
use_opt <özellik> <açık-olma-durumu> <kapalı-olma-durumu>
```

use ise yine özelliğin açık olup olmama durumunu belirtir fakat karşılığında çıktı üretmek yerine **if** ile kullanılır. Kullanımı şu şekildedir:

```
if use <özellik> ; then
    <açık-olma-durumu>
else
    <kapalı-olma-durumu>
fi
```

Özellik açma

Özellikler **USE** çevresel değişkeni ile veya **--use** parametresi veya ayar dosyasında belirtilir.

```
# --use=xxx yöntemi
$ ymp build ./repo/stuff-package --use="foo bar"
# USE=xxx yöntemi
$ USE="foo bar" ymp build ./repo/stuff-package
```

Eğer özellik listesi olarak **all** belirtirseniz **uses** dizisindeki tüm özellikler, **extra** belirtirseniz ise **uses_extra** dizisinin tümü kullanılır.

Not: Use flag sadece kaynak paketler ve derleme talimatları için kullanılabilir.

Not: sistemimizin mimarisi ile aynı adda use flag otomatik olarak tanımlanır ve kullanılır. Bu sayede tek bir ympbuild dosyası ile birden çok mimariye uyumlu paket üretilebilir.

Git tabanlı paketler

Ymp paket deposu dışında bir git deposunu kullanarak paketler oluşturmanıza izin verir. Bu sayede bir paketi depoya bağlı kalmadan paylaşabilirsiniz.

Bunun için öncelikle **ymp template** kullanarak ympbuild dosyamızı oluşturalım. Ardından oluşturduğumuz dizini git deposu haline getirelim.

```
$ ymp template --name=example --output=test-package ...
$ cd test-package
$ git init
```

Git adresimizi ekleyelim ve commit oluşturup gönderelim.

```
$ git remote add origin git@example.org:yourname/test-package.git
$ git commit -m "first commit"
$ git push -u origin master
```

Not: **ympbuild** dosyamız git deposunun ana dizininde bulunmalıdır.

Paketi aşağıdaki gibi derleyebiliriz.

```
$ ymp build --output=/path/to/output git@example.org:yourname/test-package.git
```

Not: git tabanlı paketler güvenilir olmayan paket olarak işaretlenir. Yüklemek için **--unsafe** parametresi kullanmanız gerekebilir.

Depo oluşturma

Gpg anahtarı oluşturma

Eğer yoksa gpg anahtarımızı oluşturalım.

```
gpg --generate-key
```

Gpg anahtarının repicent değerini bulma

Şimdi Repicent değerimizi bulalım.

```
gpg --list-keys
```

Depo için dizin oluşturma

Depo olarak kullanılacak dizinimi oluşturalım. İndex almak için aşağıdaki gibi bir betik yazalım.

```
ymp repo --index ./ \
    --move --name="main" \
    --gpg:repicent=<repicent-değeri> \
    --verbose
# --move paketleri taşır
# --verbose detaylı çıktı verir
# --name depo adını belirtir
# --gpg:repicent değeri belirtilen anahtarı kullanır.
```

Not: **--gpg:repicent** parametresinin değerini açıktan yazmak yerine **/etc/ymp.yaml** dosyası içine yazabilirsiniz.

Paketlerimizi dizine kopyalayıp index alacak olan betiği çalıştırıbiliriz.

Depoyu yayınlama

Depoyu bir http server uygulaması kullanarak veya ymp ile yayınlaabilirsiniz. Örneğin busybox httpd için:

```
busybox httpd -p <port> -h <depo-dizini> -v -f
# -v detaylı çıktı için
# -f arkada çalışmasını önlemek için
```

Python http.server kütüphanesi ile

```
python3 -m http.server <port>
```

ymp ile

```
ymp httpd --port=<port> --unblock
# --unblock ctrl-c kullanarak kapatabilmek için.
```

Depoyu ekleme

Yayına aldığımız depo içerisindeki ymp-index.yaml dosyasının adresini alın. **ymp-index.yaml** yerine **$uri** gelecek şekilde değiştirin. Elde ettiğiniz adresi depo olarak ekleyin. Örneğin aşağıdaki gibi bir adres için

```
# Şu bağlantı için şu şekilde eklenir.
http://10.0.0.2:8000/ymp-index.yaml
# Şu şekilde eklenir.
```

```
ymp repo --add --name=<depo-adı> 'http://10.0.0.2:8000/$uri'
# Gpg anahtarını ekleyelim.
ymp key --add 'http://10.0.0.2:8000/ymp-index.yaml.asc'
```

Not: Terminalde **$** işareti çift tırnak **"** içerisinde değişken değeri ifade eder. Bu sebeple tek tırnak **'** işareti içerisinde yazmalısınız.

Son olarak depo indexi güncelleyelim.

```
ymp repo --update
```

Sysconf Dosyaları

Sysconf dosyaları, ymp paket yöneticisi için hook mekanizması olarak kullanılır. Sysconf dosyaları /etc/sysconf.d dizininde bulunur. Bunları *ymp sysconf* komutunu kullanarak manuel olarak tetikleyebilirsiniz.

Örnek sysconf betiği:

```
current=$(date +%s -r /usr/share/glib-2.0/schemas/)
last=$(cat /var/lib/ymp/sysconf/glib/update.date)
if [ "$current" != "$last" ] ; then
    glib-compile-schemas /usr/share/glib-2.0/schemas/
    date +%s -r /usr/share/glib-2.0/schemas/ > /var/lib/ymp/sysconf/glib/update.date
fi
```

Dizin değiştirme zamanını kullanarak işlemi kontrol edebilirsiniz.

- *OPERATION* çevresel değişkeni mevcut işlemi belirtir (örneğin, postinst, postrm, preinst).
- *OPERATION* değişkeni iletilir, ancak diğer çevresel değişkenler kaldırılır.
- *OPERATION* değişkeni olarak kurulum ve kaldırma için postinst preinst postrm prerm bulunur. Diğer eylemlerde ile işlemle aynı ada sabiptir.
- Sysconf dizinleri (*/var/lib/ymp/sysconf/<hookname>*) ymp tarafından oluşturulur ve paket kaldırıldıktan sonra kaldırılır.
- Önemli bir not olarak, hedef paketten *preinst* ve *postrm* kullanılamaz çünkü paket sysconf dosyaları paketin bir parçasıdır.

Bu yapı, sysconf dosyaları aracılığıyla paket yapılandırmalarının esnek özelleştirmeleri ve yönetimi için olanak sağlar.

Sistem

Servis Yönetimi

OpenRC

Türkmen varsayılan olarak **openrc** kullanır.

Kurulum

ymp kullanarak openrc yüklemek için aşağıdaki komutu kullanabilirsiniz:

```
$ ymp install openrc
```

Kaynak koddan derlemek için aşağıdaki adımları izlemelisiniz:

```
$ git clone https://github.com/OpenRC/openrc
$ cd openrc
$ meson setup build --prefix=/usr
$ ninja -C build install
```

Basit kullanım

Servis etkinleştirip devre dışı hale getirmek için **rc-update** komutu kullanılır.

```
# servis etkinleştirmek için
$ rc-update add udhcpc boot
# servisi devre dışı yapmak için
$ rc-update del udhcpc boot
# Burada udhcpc servis adı boot ise runlevel adıdır.
```

Servisleri başlatıp durdurmak için ise **rc-service** komutu kullanılır.

```
$ rc-service udhcpc start
# veya şu şekilde de çalıştırılabilir.
$ /etc/init.d/udhcpc start
```

Servislerin durmunu öğrenmek için **rc-status** komutu kullanılır. Ayrıca sistemdeki servislerin sonraki açılışta hangisinin başlatılacağını öğrenmek için ise parametresiz olarak **rc-update** kullanabilirsiniz.

```
# şu an hangi servislerin çalıştığını gösterir
$ rc-status
# sonraki açılışta hangi servislerin çalışacağını gösterir
$ rc-update
```

Sistemi kapatmak veya yeniden başlatmak için **openrc-shutdown** komutunu kullanabilirsiniz.

```
# kapatmak için
$ openrc-shutdown -p 0
# yeniden başlatmak için
$ openrc-shutdown -r 0
```

Docker içinde servis çalıştırma

Docker veya farklı bir ortamda sistem başlatılmadığı için servisler normal olarak çalıştırılmayacaktır. Fakat aşağıdaki adımları uygulayarak servis başlatmamız mümkündür.

```
# Önce /run/openrc dizini oluşturulur
$ mkdir -p /run/openrc
# Ardından boş softlevel dosyası oluşturulur
$ touch /run/openrc/softlevel
```

Bu işlemden sonra servis başlatmamız mümkün hale gelmektedir. Servisi aşağıdaki komut ile başlatabiliriz.

```
$ rc-service sshd start
```

Servis dosyası

Openrc servis dosyaları basit birer **bash** betiğidir. Bu betikler **openrc-run** komutu ile çalıştırılır ve çeşitli fonksiyonlardan oluşabilir. Servis dosyaları **/etc/init.d** içerisinde bulunur. Servisleri ayarlamak için ise **/etc/conf.d** içerisine aynı isimle ayar dosyası oluşturabiliriz.

Çalıştırılacak komut komut parametreleri ve **pidfile** dosyamızı aşağıdaki gibi belirtebiliriz.

```
description="Ornek servis"
command=/usr/bin/ornek-servis
command_args=--parametre
pidfile=/run/ornek-servis.pid
```

Bununla birlikte **start**, **stop**, **status**, **reload**, **start_pre**, **stop_pre** gibi fonksiyonlar da yazabiliriz.

```
...
start(){
    ebegin "Starting ${RC_SVCNAME}"
    start-stop-daemon --start --pidfile "/run/servis.pid" --exec /usr/bin/ornek-servis --parametre
}
...
```

Servis bağımlılıklarını belirtmek için ise **depend** fonksiyonu kullanılır.

```
...
depend() {
  need localmount
  after dbus
}
...
```

Openrc teorik olarak sysv-init betiklerini de çalıştırabilir. Fakat kesinlikle tavsiye edilmemektedir.

İnit

Sistemi başlatmak için herhangi bir servis yöneticisi kullanmak zorunda değilsiniz. Bunun yerine kendiniz bir betik yardımı ile başlatabilirsiniz.

Bunun için **/sbin/init** dosyası oluşturmalı ve içine açılışta çalışacak komutları yazmalısınız.

Bu dosyada sırası ile yapılması gereken işlemler şu şekildedir:

- Gereken çekirdek modülleri yüklenir.
- Dosya sistemi bağlanır.
- Gereken servisler başlatılır (dbus network-manager lightdm gibi)
- /dev/tty* ile giriş yapılması için *getty* çalıştırılır.

Modüllerin yüklenmesi

Varsa systemd-udev veya eudev gibi araçlar kullanılır.

```
# systemd-udevd için
/lib/systemd/systemd-udevd &
udevadm trigger -c add
udevadm settle
# eudev için
udevd &
udevadm trigger -c add
udevadm settle
```

Bunun dışında elle yüklemek mümkündür.

```
find /sys/ -name modalias | xargs sort -u | xargs -n 1 modprobe
```

Dosya sisteminin bağlanması

Basitçe şu şekilde bağlayabilirsiniz.

```
mount -a
```

Veya **/etc/fstab** dosyasını okuyup gerekli işlemleri gerçekleştirebilirsiniz.

Servislerin başlatılması

Servisler arkada çalıştırılacağı için init betiğinizi blocklamaması gerekmektedir. Örneğin şu şekilde çalıştırabiliriniz.

```
dbus-daemon --system &
NetworkManager &
lightdm &
```

Not: Çıktıları bir dosyaya yönlendirmek hata ayıklamak ve karmaşayı engellemek için daha iyi olabilir.

Getty başlatılması

Sisteme giriş yapılabilmesi için /dev/tty* için getty başlatılması gerekmektedir.

```
# Örnğin /dev/tty1 için
/sbin/getty 115200 tty1 &
```

Not: Oturumdan çıkıldığında getty kapanacağı için tekrar başlatmanız gerekebilir.

Dikkat edilmesi gerekenler

- İnit pid 1 olarak çalıştığı için asla kapatılmamalıdır. Kapanmayı engellemek için **exec sleep inf** kullanabilirsiniz.
- reboot / poweroff / shutdown komutları servis yöneticilerine ait olduğu için (sysvinit gibi) özel init ile çalışmaz. Bu komutlara **-f** parametresi ekleyebilir veya **/proc/sysrq-trigger** kullanabilirsiniz.
- Özel yazılmış initte servis başlat durdur yeniden başlat gibi özellikler bulunmadığı için elle kontrol sağlamalısınız.

Ağ ve İnternet

udhcpc

Busybox tarafından sağlanan dhcp client uygulamasıdır.

Kullanımı aşağıdaki gibidir.

Öncelikle ağ arayüzünü etkinleştirelim.

```
# burada eth0 ağ arayüzü adıdır.
# ağ arayüzleri listesine /sys/class/net/ içerisinden ulaşabiliriniz.
$ ip link set eth0 up
```

Şimdi de bağlantı kuralım.

```
# -s ile belirtilen betiğin varsayılan konumu /usr/share/udhcpc/default.script
$ udhcpc -i eth0 -s udhcpc.sh
```

udhcpc çalışırken bir betiğe ihtiyaç duyar. Bu betik sayesinde ağ bağlantısı kurulur.

Betik içeriği aşağıdaki gibi olmalıdır.

```
#!/bin/sh
ip addr add $ip/$mask dev $interface
if [ "$router" ] ; then
  ip route add default via $router dev $interface
fi
```

Burada değişkenler şu şekildedir.

- **$ip**: ip adresi
- **$mask**: netmask değeri
- **$interface**: ağ donanımı adı
- **$route**: route adresi (tanımlı olmayabilir)

OpenRC Servisi

Türkmen **udhcpc** için **openrc** servisi sağlar. Bu servisini etkinleştirmek için aşağıdaki komutu kullanabilirsiniz. Bu sayede sistem açıldığında ağ bağlantınız udhcpc ile kurulmuş olur.

```
rc-update add udhcpc boot
```

Not: Yeni bir aygıt taktığınızda otomatik olarak bağlantı kurulmaz. Bunun için servisi yeniden başlatmanız gerekebilir.

```
rc-service udhcpc restart
```

NetworkManager

networkmanager paketi tarafından sağlanan ağ yönetim uygulamasıdır.

Kurulumu

ymp ile kurmak için aşağıdaki komutu kullanabilirsiniz.

```
$ ymp install networkmanager
```

Kaynak koddan derlemek için aşağıdaki işlemleri uygulayabilirsiniz.

```
# Tüm seçenekler için meson_options.txt dosyasına bakın
$ meson setup build \
    -Dsystemd_journal=false \
    -Dsystemdsystemunitdir=no \
    -Dsession_tracking=no \
    -Dsession_tracking_consolekit=false ...
$ ninja -C build
$ ninja -C build install
```

Openrc servisi

Türkmen networkmanager için openrc servisi sağlar. Bu sayede ağ bağlantısı sistem açıldığında otomatik olarak yapılmış olur. Servisi aşağıdaki gibi etkinleştirebiliriz.

```
$ rc-update add networkmanager
```

Servisi elle başlatma

NetworkManager komutu kullanılarak servis elle başlatılabilir. Bunun için önce **dbus** etkinleştirilmelidir. Eğer **-d** parametresi ile başlatırsanız servis hata ayıklama modunda başlatılacaktır.

```
# dbus servisini açmak için
$ rc-service dbus start
# openrc kullanmadan çalıştırabiliriz.
$ mkdir -p /var/run/dbus
$ dbus-daemon --system &
# servisi başlatma
$ NetworkManager
```

Wifi

Wifi bağlantısı sağlamak için wpa-supplicant kullanılır. Ayrıca wifi kartının sürücüsü için linux-firmware gerekmektedir.

Wpa-supplicant kurulumu

Öncelikle aşağıdaki gibi gerekli paketi yükleyelim.

```
$ ymp install wpa_supplicant
```

Ardından servisini başlatalım.

```
# servisi başlangıca ekleyelim
$ rc-update add wpa_supplicant
# servisi başlatalım
$ rc-service wpa_supplicant start
```

Bağlantının kurulması

Öncelikle wifi kartımızın donanım adını aşağıdaki komut ile tespit edelim.

```
$ ip addr
...
3: wlan0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen 1000
   link/ether 8e:d2:82:0e:96:41 brd ff:ff:ff:ff:ff:ff permaddr 08:5b:d6:0c:45:bd
...
```

Daha sonra wifi kartımızı çalıştıralım.

```
$ ip link set wlan0 up
```

Ardından bağlantı için ayar dosyasını oluşturalım.

```
$ wpa_passphrase 'SSID' 'parola' > /etc/wpa_supplicant/wpa_supplicant.conf
```

Daha sonra **wpa_supplicant** komutunu kullanarak ayar dosyasına göre bağlantı sağlayalım.

```
# -B arkada çalışması için
# -i donanım adını belirtmek için
# -c ayar dosyası konumu için
$ wpa_supplicant -B -i wlan0 -c /etc/wpa_supplicant/wpa_supplicant.conf
```

Bağlantı kurulduktan sonra ip adresi almak için udhcpc kullanalım.

```
$ udhcpc -i wlan0
```

Network Manager ile kullanılması

wpa_supplicant servisini etkinleştirdikten sonra networkmanager servisini yeniden başlatalım.

```
$ rc-service networkmanager restart
```

Ardından wifi donanımımızı açalım.

```
$ nmcli radio wifi on
```

Ardından wifi ağlarını listeleyelim.

```
$ nmcli device wifi list
```

Ardından wifi ağına bağlanalım.

```
# Parolayı sizden yazı girdi olarak sorması için --ask kullanılır.
$ nmcli device wifi connect "SSID" --ask
# Parolayı doğrudan komuta ekleyebilirsiniz
$ nmcli device wifi connect "@" password "password"
```

Bluetooth

Bluetooth bağlantısı sağlamak için bluez kullanılır. Ayrıca bluetooth kartının sürücüsü için linux-firmware gerekmektedir.

Kurulum

Öncelikle **bluez** kurulumu yapalım.

```
$ ymp install bluez
```

Ardından servisi etkinleştirelim.

```
$ rc-update add bluetooth
$ rc-service bluetooth start
```

Bağlantının kurulması

İlk olarak klavye mouse gibi aygıtları bağlayabilmek için gerekli modülü yükleyelim.

```
modprobe uhid
```

Bunu açilişta yüklemek için aşağıdaki gibi ekleme yapabilirsiniz.

```
echo uhid > /etc/modules-load.d/bluetooth.conf
```

bluetoothctl komutu ile bağlantı kurmamız mümkündür. Öncelikle kullanılabilir aygıtları görüntüleyelim.

```
$ bluetoothctl scan on
...
[NEW] Device 1A:AA:D4:9C:8D:F5 Xiaomi Mi 6X
...
```

Arama modundan **ctrl+c** ile çıkalım. Çıktıdan bağlanmak istediğimiz aygıtın adresini alalım ve aşağıdaki gibi eşleştirelim.

```
$ bluetoothctl pair 1A:AA:D4:9C:8D:F5
```

Şimdi de bağlanalım.

```
$ bluetoothctl connect 1A:AA:D4:9C:8D:F5
```

Not: Özellikle dual boot kullanıyorsanız donanım sürekli olarak farklı şekilde haberleşmeye çalıştığı için bağlanma sorunları oluşabilir. Bu durumun üstesinden gelmenin en basit yolu aygıtı kaldırıp tekrar eklemektir.

Otomatik bağlanma

Her seferinde bağlantıyı elle yapmayıp aygıta güvenmek için aşağıdaki komutu uygulayabilirsiniz.

```
$ bluetoothctl trust 1A:AA:D4:9C:8D:F5
```

Aygıtı kaldırma

Bağlantıyı kesmek için öncelikle tanınan aygıt listesine bakalım.

```
$ bluetoothctl devices
```

Şimdi aygıtın bağlantısını keselim.

```
$ bluetoothctl disconnect 1A:AA:D4:9C:8D:F5
```

Ardından aygıtı silelim.

```
$ bluetoothctl remove 1A:AA:D4:9C:8D:F5
```

Ssl sertifikası yükleme

Eğer elimizde .crt uzantılı bir dosya varsa ve bunu metin düzenleyici ile açamıyorsak aşağıdaki gibi .pem uzantılı hale çevirelim:

```
$ openssl x509 -in ./my-ssl-cert.crt -out ./my-ssl-cert.pem -outform PEM
```

İlk olarak elimizde bulunan ssl sertifikası (.pem uzantılı) **/usr/share/ca-certificates/custom** dizini oluşturup içine kopyalayalım.

```
$ mkdir -p /usr/share/ca-certificates/custom/
$ cp ./my-ssl-cert.pem /usr/share/ca-certificates/custom/my-ssl-cert.pem
```

Ardından ssl veritabanını güncelleyelim.

```
$ update-ca-certificates
```

Ardından eğer varsa bundle dosyasına ekleyelim.

```
$ echo "My SSl Cert" >> /etc/ssl/cert.pem
$ cat ./my-ssl-cert.pem >> /etc/ssl/cert.pem
```

Firefox için yükleme

Firefox ssl sertifikasını sistem üzerinden kullanmak yerine kendi içerisinde bulunan sertifikalara bakar. Bu sebeple firefox için de elle yüklemek gerekli olabilir.

Bunun için sırasıyla:

```
-> Ayarlar
-> Gizlilik ve Güvenlik
-> Sertifikalar
```

```
-> Sertifikaları göster...
-> Makamlar
-> İçe aktar...
```

Kısmından .pem uzantılı dosyanızı güven kutucuklarını işaretleyip ekleyin.

Oturum ve Başlangıç

Kullanıcılar

Kullanıcı yönetimi

Her kullanıcının kendisine ait bir uid ve gid değeri bulunur. Bu değer sistem kullanıcıları için 1000 den küçük normal kullanıcılar için ise 1000 ve daha büyük bir değere sahiptir.

Not: uid değeri 0 ise kullanıcı tam yetkilidir. (root yetkisi)

Bu uid değerleri **/etc/passwd** dosyası içerisinde bulunur.

```
root:x:0:0:root:/root:/bin/ash
pingu:x:1000:1000:Linux User:/home/pingu:/bin/bash
# pingu : kullanıcı adı
# x : bir anlamı yok
# 1000 : uid değeri
# 1000 : gid değeri
# Linux User : kullanıcının gözüken adı
# /home/pingu : kullanıcı ev dizini
# /bin/bash : kullanıcı kabuğu
```

Not: Türkmen diğer dağıtımlardan farklı olarak **/home** yerine **/data/user** dizini kullanır. Bu yüzden kullancı ev dizinini uygun olarak ayarlamanız gerekir.

Kullanıcı ekleme ve silme

Kullanıcı ekleme

Yeni kullanıcı eklemek için **useradd** veya **adduser** komutu kullanılır.

```
$ useradd -d /data/user/pingu -m -u 1000 -g 1000 -s /bin/bash pingu
# -d ev dizini
# -m ev dizini oluşturmak için (yoksa)
# -u uid değeri
# -g gid değeri
# -s varsayılan kabuk
$ adduser -D -h /data/user/pingu -u 1000 -s /bin/bash pingu
# -D oluştururken kullanıcı parolası sormaması için
# -h ev dizini konumu
# -u uid değeri
# -s varsayılan kabuk
$ echo "pingu:x:1000:1000:Linux User:/data/user/pingu:/bin/bash" >> /etc/passwd
$ mkdir -p /data/user/pingu
$ chown pingu /data/user/pingu
# Bu şekilde de kullanıcı ekleyebilirsiniz fakat tavsiye edilmez.
```

Sysconf yardımı ile kullanıcı ekleme

Türkmen içerisinde **/etc/passwd.d** dizini bulunur. Bu dizin sayesinde paketler kurulurken kullanıcı eklemek mümkün olur. Bunun için **/etc/passwd.d/paketadı** dosyası oluşturup içerisine passwd dosyasına eklenmesi gereken satırlar yazılabilir. Bu sayede paket kurulurken sysconf bu dosyayı algılayarak kullanıcıyı otomatik olarak sisteme dahil eder.

Kullanıcı silme

Kullanıcı silmek için **userdel** veya **deluser** komutu kullanılır.

```
$ userdel -r pingu
# -r ev dizinini silmek için
$ deluser --remove-home pingu
# --remove-home ev dizinini silmek için
$ sed -i "/^pingu:x:.*/d" /etc/passwd
$ rm -rf /data/user/pingu
# Bu şekilde de kullanıcı silebilirsiniz fakat tavsiye edilmez.
```

Parola ayarlama

Kullanıcılar arası geçiş yapmak için **su** komutu kullanılır. Bu komut geçilecek olan kullanıcının parolasını sorar. Bu yüzden kullanıcılara bir parola tanımlamamız gerekmektedir.

Parola tanımlamak için **passwd** komutu kullanılır.

```
$ passwd pingu
# kullanıcı adı belirtmezseniz root kabul edilir.
```

Parola ayarlamanın diğer bir yolu ise **usermod** komutu kullanmaktır. Bu yöntemde **openssl** komutundan yararlanılır.

```
# önce hash elde edelim
$ openssl passwd -6 'parola'
-> $6$GBPcPGqQLyLcYkKl$1z5BOQB36E31.VIJyGJXwCc6invR2WgeaSI9Jz7QZU/QZbffEm.J8edQkyIBtRWpSa.VFob3p/BH84Unag1Y60
# -6 sha512 formatında hash üretmek için.
# elde ettiğimiz değer ile parola belirleyelim.
$ usermod -p <hash-değeri> pingu
# Şu şekilde de tanımlayabilirsiniz.
$ usermod -p "$(openssl passwd -6 'parola')" pingu
```

Not: Özel karakterler ile parola oluşturma durumuna karşı tek tıknak (') işareti içerisine yazmanız gerekmektedir.

Not: Parolanın kabuğun history bölümünde gözükmesi güvenlik sorunlarına sebep olabilir. İşlem bittikten sonra history dosyasını temizlemenizi öneririm.

Kullanıcılar arası geçiş

Root yetkisi

Root yetkisi sistemde tam erişime sahip yetki düzeyidir. Bu yetki sayesinde sistemde değişiklik yapılabilmektedir. Örneğin paket kurulumu ve kaldırma gibi işlemler için root yetkisine ihtiyacımız vardır.

Root yetkisi **root** kullanıcısına aittir. Bu kullanıcının **UID** ve **GID** değeri 0dır. Ev dizini ise **/root** dizinidir.

Root yetkisinin alınması

Kullanıcı değiştirmek için **su** komutu kullanılır. Eğer bu komuta parametre vermezseniz **root** kullanıcısına geçiş yapılmış olur.

```
$ su
-> Password:
$ id
-> uid=0(root) gid=0(root) groups=0(root)...
```

suid kavramı

su komutunun çalışabilmesi için **suid** iznine sahip olması gereklidir. Bunu aşağıdaki gibi kontrol edebiliriz.

```
# s harfi suid iznini ifade eder.
$ ls -la /bin/su
-> -rws--x--x 1 root root 72816 Jan 14 10:25 /bin/su
```

suid izni vermek için **chmod u+s** izni geri almak için ise **chmod u-s** komutu kullanılır. Bu komutlar sadece root kullanıcısı tarafından çalıştırılabilir.

```
# yetki vermek için
$ chmod u+s /bin/su
# yetkiyi geri almak için
$ chmod u-s /bin/su
```

setuid sistem çağrısı

suid izni verilen dosyalar **setuid()** ve **setgid()** sistem çağrısını kullanabilirler. Örneğin aşağıdaki gibi bir C kodumuz olsun.

```
#include <unistd.h>
#include <stdlib.h>
int main(){
    setuid(0);
    setenv("USER","root",1);
    return system("sh");
}
```

Bu C kodunu gcc ile derleyelim ve **suid** izni verelim. **suid** yalnızca root kullanıcısı ayarlayabileceği için bu işlem root kullanıcısı ile yapılmalıdır.

```
$ gcc -o main.c main
$ chmod u+s main
```

Derlenen ve **suid** izni ayarlanan dosyamızı normal kullanıcımız ile çalıştırdığımızda root yetkisi alacaktır. **su** komutumuz bundan yararlanarak çalışmaktadır.

Not: suid iznine sahip dosyalar potansiyel güvenlik açığı oluşturabilir.

suid engelleme

Dosya sisteminde **suid** iznini engellemek için **nosuid** seçeneği etkinleştirilebilir. **/etc/fstab** dosyamızda ilgisi satır şu şekilde olabilir.

```
...
# filesystem     mountpoint   type   options           dump/pass
/dev/nvme0n1p2   /            ext4   default,rw,nosuid 0 0
...
```

busybox su

Busybox bize **su** komutu sağlayabilmektedir. Bu komutu kullanmak için öncelikle busyboxun kopyası oluşturulmalı ve ona **suid** yetkisi verilmelidir. Türkmen varsayılan su komutu olarak busyboxu kullanmaktadır.

```
$ install /bin/busybox /bin/su
$ chmod u+s /bin/su
```

Kullanıcı Kabuğu

Kullanıcı oturum açtığı anda kullanıcının varsayılan kabuk uygulaması (shell) başlatılır.

Bu varsayılan kobuk konumu **/etc/passwd** dosyasında belirtilmiştir.

```
pingu:x:1000:1000::/data/user/pingu:/bin/ash
```

Burada **/bin/ash** kabuk konumudur.

Kabuğu değiştirme

Öncelikle değiştirmek istediğiniz kabuğun konumunu **/etc/shells** içerisine eklemeniz gerekmektedir. Bu işlemi yapmazsanız kullanıcıyla giriş yapamazsınız.

Daha sonra **/etc/passwd** dosyasından kabuğun konumunu değiştirmemiz gerekir.

Not: Kabuk konumu parametre alamaz ve tam konum olmak zorundardır.

Kabuk değiştirildikten sonra tekrar giriş yapmanız gerekebilir.

Eğer kabuk konumu olarak **/sbin/nologin** kullanırsanız kullanıcının giriş yapmasını engellemiş olursunuz. Bu genellikle servislerin oluşturduğu kullanıcılar için kullanılır.

Unix Kabuğu

/bin/sh sistem tarafından kullanılan genel kabuktur. Bu kabuk debian tabanlılar için **/bin/dash**, alpine linux ve türkmen linux için **/bin/busybox**, diğerleri için **/bin/bash** konumuna sembolik bağlıdır.

Bu kabuk sistem açılırken kullanılır. Aşağıdaki gibi bir C kodu ile durumu örnekleyebiliriz.

```c
#include <stdio.h>
int main(){
    system("echo $0");
    return 0;
}
```

Bu kod çalıştırıldığında ekrana **sh** yazacakdır. Çünkü system komutu şununla eşdeğer şekilde çalışır.

```c
execl("/bin/sh", "sh", "-c", command, (char *) NULL);
```

Bu yüzden /bin/sh kabuğunu iyi seçmek sistem tasarımı açısından önemli olabilir. Farklı unix kabukları ve avantaj/dezavantajları aşağıdaki gibi özetlenebilir.

- /bin/dash : Debian tarafandan geliştirilir. Sadece kullanıcılar tarafından fakat yazılımlar tarafından ihtiyaç duyulmayan ek özellikler bulunmaz. (Tab tuşu ile tamamlama gibi). Bu sayede küçük boyutludur ve hızlı çalışır. Bash uyumlu değildir.
- /bin/ash : Busybox tarafından sağlanır. Ek pakete gereksinim duymaz. Basit seviyede özelliklere sahiptir. Kısmen bash uyumludur.
- /bin/bash : Bash GNU/Linux dağıtımlarının genelinde varsayılandır ve Çok az sorun çıkarır.

Unix kabuğunu değiştirmek için ayağıdaki gibi bir yol izleyebilirsiniz.

```
$ rm -f /bin/sh
$ ln -s bash /bin/sh
```

Oturumlar

Elogind

elogind systemd projesinde bulunan **logind** uygulamasının systemd'den bağımsız çalışabilen halidir. Genellikle systemd'i kullanmayan ancak KDE veya GNOME gibi systemd'e bağımlı yazılımları kullanmak isteyen kullanıcılar için tercih edilir. Amacı kullanıcı oturumlarını yönetmektir. **Pam** kullanarak çalışır.

Kurulumu

Ymp kullanarak aşağıdaki gibi kurulum yapabilirsiniz.

```
$ ymp install elogind
```

Veya kaynak koddan derlemek için aşağıdaki komutları kullanabilirsiniz.

```
# Seçenekler için meson_options.txt dosyasına bakın.
$ meson setup build -Dpam=true ...
$ ninja -C build
$ ninja -C build install
```

Ardından openrc servisini etkinleştirelim. Bunun için aşağıdaki komuttan yararlanabiliriz.

```
$ rc-update add elogind
```

elogind **pam** ile çalıştığı için pam yapılandırmasına eklememiz gerekmektedir. Bunun için **/etc/pam.d/system-auth** dosyasına aşağıdaki satırı ekleyelim.

```
# /etc/pam.d/system-auth dosyası içine en alta ekleyin.
session    include    elogind-user
```

agetty servisinin ayar dosyasında **login** komutu ayarlamak gerekebilir. Bunun sebebi **login** komutunun varsayılan olarak pam kullanmadan çalışan busybox tarafından sağlanmasıdır. Bunun için **/etc/conf.d/agetty** içerisini aşağıdaki gibi değiştirelim.

```
...
agetty_options="-l /usr/bin/login"
...
```

libsystemd isteyen uygulama ve kütüphaneler ile uyumluluğu arttırmak için sembolik bağ atabilirsiniz. (İsteğe bağlı)

```
ln -s libelogind.so.0 /lib/libsystemd.so.0
ln -s libelogind.so.0 /lib/libsystemd.so.0.35.0
ln -s pam_elogind.so /lib/security/pam_systemd.so
ln -s libelogind.pc /lib/pkgconfig/libsystemd.pc
ln -s libelogind.pc /lib/pkgconfig/libsystemd-login.pc
ln -s elogind /usr/include/systemd
```

Benzer şekilde **systemd** çalışıyormuş gibi algımasını sağlamak için servis dosyasına aşağıdaki gibi ekleme yapabilirsiniz.

```
...
start_pre() {
  # systemd is running stuff
  checkpath --mode 0755 -d /run/systemd/
  checkpath --mode 0755 -d /run/systemd/system/
}
...
```

Oturumların listelenmesi

Oturum listelemek için **loginctl** komutunu kullanabilirsiniz. Bu komut aşağıdaki gibi çıktı verir.

```
SESSION UID USER SEAT  TTY
      1   0 root seat0 tty1

1 sessions listed.
```

Kullanım

loginctl komutu, oturum yöneticisini denetlemek ve analiz etmek için kullanılır. Örneğin, sistemi kapatmak veya yeniden başlatmak için şu komutları kullanabilirsiniz:

```
loginctl poweroff
loginctl reboot
```

Uyku moduna almak için ise şu komutları kullanabilirsiniz:

```
loginctl suspend
```

Not: Uyku modu bazı donanımlarda düzgün çalışmayabilir.

Oturum kontrolü

loginctl komutu, Linux sistemlerinde oturumları kontrol etmek ve yönetmek için kullanılır. Bu komut, kullanıcıların oturumlarını listeleme, oturumları kapatma, ekranları kilitleme, sistem işlemlerini gerçekleştirme gibi çeşitli işlemleri gerçekleştirmek için kullanılır.

Oturumları Listeleme:

```
loginctl list-sessions
```

Bu komut, sistemdeki tüm oturumları listeler. Her oturumun bir oturum kimliği (session id) bulunur.

Oturumu Kapatma:

```
loginctl terminate-session <session-id>
```

Bu komut, belirtilen oturumu sonlandırır. <session-id>, sonlandırmak istediğiniz oturumun kimliğidir.

Ekranı Kilitleme:

```
loginctl lock-session <session-id>
```

Bu komut, belirtilen oturumun ekranını kilitlemeye yarar. <session-id>, kilitlemek istediğiniz oturumun kimliğidir.

Ekran Kilidini Açma:

```
loginctl unlock-session <session-id>
```

Bu komut, belirtilen oturumun ekranının kilidini açar. <session-id>, kilidini açmak istediğiniz oturumun kimliğidir.

Yetkilendirme

Polkit

Polkit yetkisiz uygulamaların yetkili olan uygulamalar ile iletişim kurmasını sağlayan bir araçtır. Örneğin gparted gibi uygulamaları başlatmak için **pkexec** komutu ile root yetkisi sağlanır.

Kurulum

ymp ile **polkit** yüklemek için aşağıdaki komut kullanılır:

```
$ ymp install polkit
```

Ardından polkitin servisi için gereken dosyalara **suid** yetkisi verelim.

```
$ chmod 4755 /usr/bin/pkexec
$ chmod 4755 /usr/lib/polkit-1/polkit-agent-helper-1
```

Sonra servisi etkinleştirelim.

```
$ rc-update add polkit # açılışta başlaması için
$ rc-service polkit start # başlatmak için
```

Son olarak **/etc/shells** dosyasını kontrol edelim. **echo $SHELL** komutunu çalıştırınca çıkan değerin bu dosyada bulunması gerekir.

Polkit-agent

Kullandığınız masaüstüne göre polkit-agent yüklemeniz gereklidir. Aşağıda tablo olarak gerekenler verilmiştir.

Polkit Agent Listesi

Masaüstü	paket	sağladığı komut
Kde	polkit-kde-agent	/usr/lib64/libexec/polkit-kde-authentication-agent-1
lxde	lxsession	/usr/bin/lxpolkit
cinnamon	polkit-gnome	/usr/libexec/polkit-gnome-authentication-agent-1
xfce	xfce-polkit	/usr/libexec/xfce-polkit

Not: polkit agent desdeği olmayan masaüstülerde **polkit-gnome** kullanabilirsiniz.

Kurallar

Polkit kural dosyaları **/usr/share/polkit/rules.d/** ve **/etc/polkit/rules.d** dizinlerinde bulunur. Bu dizinlere kurallar ekleyebilirsiniz. Örneğin test gurubundaki herşeye yetki vermek için :

```
polkit.addRule(function(action, subject) {
    if (subject.isInGroup("test")) {
      return polkit.Result.YES;
  }
});
```

Not: Yukarıdaki örnek sadece konuyu anlatmak içindir. Güvenlik sorunlarına sebep olabilir.

Eylemler

Kural dosyalarına benzer şekilde eylem dosyaları da bulunur. Bu dosyalar ise **/usr/share/polkit-1/actions** içerisinde bulunur.

Örneğin:

```
<policyconfig>
  <vendor>Test Vendor</vendor>
  <vendor_url>https://example.org</vendor_url>
  <action id="org.turkmen.test.application">
    <description>Test application</description>
    <message>Authentication is required for test application</message>
    <message xml:lang="tr">Test uygulaması için yetkilendirme gerekiyor</message>
    <icon_name>preferences-system</icon_name>
    <defaults>
      <allow_any>yes</allow_any>
      <allow_inactive>yes</allow_inactive>
      <allow_active>yes</allow_active>
    </defaults>
    <annotate key="org.freedesktop.policykit.exec.path">/usr/sbin/test</annotate>
    <annotate key="org.freedesktop.policykit.exec.allow_gui">true</annotate>
  </action>
</policyconfig>
```

Polkit kullanımı

pkexec komutunu kullanarak polkit çalıştırabilirsiniz. Örneğin:

```
$ pkexec /bin/bash
```

Burada dikkat etmemiz gereken polkit eylemi olmayan bir komutu çalıştırdığımızda çevresel değişkenler korunmadığı için gui kullanan uygulamaları çalıştıramazsınız.

PAM

Linux sistemlerinde güvenliği sağlamak için kullanılan PAM (**Pluggable Authentication Modules**), modüler bir yapıya sahiptir ve kullanıcı kimlik doğrulama işlemlerini yönetir.

PAM, modüler bir yapıya sahiptir ve her bir modül belirli bir kimlik doğrulama görevini yerine getirir. Örnek olarak, "pam_unix" modülü, parola tabanlı kimlik doğrulamayı yönetirken, "pam_ldap" modülü LDAP sunucusu üzerinden kimlik doğrulamasını sağlar.

Yapılandırma:

PAM'ın yapılandırması genellikle **/etc/pam.d/** dizinindeki dosyalarda yapılır. Bu dosyalarda, farklı hizmetler için kimlik doğrulama adımları belirtilir. Örneğin, login, sshd, su gibi hizmetler için ayrı ayrı PAM yapılandırmaları bulunabilir.

PAM yapılandırma dosyaları, satırlar halinde okunur. Her satır, bir PAM modülünü ve bu modülün ne zaman ve nasıl kullanılacağını tanımlar.

Temel bir PAM yapılandırma dosyası şu bölümleri içerebilir:

```
auth: Kullanıcının kimlik doğrulama süreci.
account: Kullanıcının hesap durumu kontrolü.
password: Kullanıcının parola değiştirme işlemleri.
session: Oturum başlatma ve sonlandırma işlemleri.
```

PAM yapılandırma dosyalarında, kullanılacak PAM modüllerinin tanımları bulunur. Her modül, bir kütüphane dosyasına (genellikle /lib/security/ dizininde) işaret eder. Modül tanımları, modülün seçeneklerini de içerebilir.

Örnek Bir PAM Yapılandırma Dosyası (SSH için):

```
# SSH için PAM yapılandırma dosyası (/etc/pam.d/sshd)

# Kullanıcı kimlik doğrulama işlemi
auth       required     pam_sepermit.so
auth       substack     password-auth
auth       include      postlogin

# Kullanıcı hesap durumu kontrolü
account    required     pam_nologin.so
account    include      password-auth

# Parola değiştirme işlemleri
password   include      password-auth

# Oturum başlatma ve sonlandırma işlemleri
session    required     pam_selinux.so
session    required     pam_loginuid.so
session    optional     pam_keyinit.so force revoke
session    include      password-auth
session    include      postlogin
```

PAM yapılandırma dosyalarında, PAM modüllerinin davranışını belirleyen çeşitli anahtar kelimeler bulunur. İşte bu anahtar kelimelerden bazılarının anlamları:

- **required (gereken):**
  Bu anahtar kelime, ilgili PAM modülünün başarılı bir kimlik doğrulama süreci için zorunlu olduğunu belirtir. Yani, bu modül başarısız olursa, kimlik doğrulama işlemi başarısız olur ve kullanıcı erişimi reddedilir.
- **requisite (gerekli):**
  required ile benzerdir, ancak bu modül başarısız olduğunda, geri kalan modüller çalıştırılmaz ve kimlik doğrulama işlemi doğrudan başarısız olur.
- **sufficient (yeterli):**
  Bu anahtar kelime, modülün başarılı olması durumunda, kimlik doğrulama işleminin başarılı olduğunu belirtir. Eğer bu modül başarılı olursa, geri kalan modüller çalıştırılmaz.
- **optional (isteğe bağlı):**
  Bu anahtar kelime, ilgili modülün başarısız olması durumunda, kimlik doğrulama işleminin hala devam edeceğini belirtir. Yani, bu modülün başarısız olması, kimlik doğrulama işlemini doğrudan etkilemez.

Pam hata ayıklama

Öncelikle syslogd başlatmamız gerekmektedir. Ardından **/petc/pam_debug** konumuna boş bir dosya oluşturun. Artık pam ile ilgili hata ayıklama mesajlarına syslog üzerinden erişebilirsiniz.

Pam modülü yazmak

Örnek bir pam modülü aşağıdaki gibi olabilir:

```
#include <security/pam_appl.h>

PAM_EXTERN int pam_sm_authenticate(pam_handle_t *pamh, int flags, int argc, const char **argv) {
        // Kimlik doğrulama işlemlerini gerçekleştirir
        // Burada gerekli işlemler yapılacaktır
        return PAM_SUCCESS; // İşlem başarılıysa PAM_SUCCESS döner
}

PAM_EXTERN int pam_sm_setcred(pam_handle_t *pamh, int flags, int argc, const char **argv) {
        // Kullanıcı kimlik bilgilerini ayarlar
        // Gerekli işlemler yapılabilir
        return PAM_SUCCESS; // İşlem başarılıysa PAM_SUCCESS döner
}

PAM_EXTERN int pam_sm_acct_mgmt(pam_handle_t *pamh, int flags, int argc, const char **argv) {
        // Kullanıcı hesabını yönetir
        // Gerekli işlemler yapılabilir
        return PAM_SUCCESS; // İşlem başarılıysa PAM_SUCCESS döner
}

PAM_EXTERN int pam_sm_open_session(pam_handle_t *pamh, int flags, int argc, const char **argv) {
        // Kullanıcı oturumunu açar
        // Gerekli işlemler yapılabilir
        return PAM_SUCCESS; // İşlem başarılıysa PAM_SUCCESS döner
}

PAM_EXTERN int pam_sm_close_session(pam_handle_t *pamh, int flags, int argc, const char **argv) {
        // Kullanıcı oturumunu kapatır
        // Gerekli işlemler yapılabilir
```

```
        return PAM_SUCCESS; // İşlem başarılıysa PAM_SUCCESS döner
}

PAM_EXTERN int pam_sm_chauthtok(pam_handle_t *pamh, int flags, int argc, const char **argv) {
```

```
        // Kullanıcı parolasını değiştirir
        // Gerekli işlemler yapılabilir
        return PAM_SUCCESS; // İşlem başarılıysa PAM_SUCCESS döner
}
```

Modülü aşağıdaki gibi derleyebilirsiniz:

```
gcc -o pam_example.so -fPIC -shared -lpam example.c
# -lpam libpam bağlamak için
# -fPIC position independent code açmak için
# -shared kütüphane dosyası oluşturmak için
```

Pam kullanımı

Aşağıdaki C kodu ile pam kullanarak doğrulama yapabilirsiniz.

```
#include <stdio.h>
#include <stdlib.h>
#include <security/pam_appl.h>

// PAM struct yapısı
static struct pam_conv conv = {
        misc_conv,
        NULL
};

int main(int argc, char *argv[]) {
        pam_handle_t *pamh = NULL;
        int retval;
        const char *username = "test"; // Doğrulanacak kullanıcı adı

        // PAM oturumu başlat
        retval = pam_start("login", username, &conv, &pamh);

        // PAM oturumu başlatma hatası kontrolü
        if (retval != PAM_SUCCESS) {
            fprintf(stderr, "PAM initialization failed: %s\n", pam_strerror(pamh, retval));
            exit(EXIT_FAILURE);
        }

        // Kullanıcıyı doğrula
        retval = pam_authenticate(pamh, 0);

        // Kullanıcı doğrulama başarılı mı kontrolü
        if (retval == PAM_SUCCESS) {
            printf("Authentication succeeded!\n");
        } else {
            fprintf(stderr, "Authentication failed: %s\n", pam_strerror(pamh, retval));
        }

        // PAM oturumu sonlandır
        if (pam_end(pamh, retval) != PAM_SUCCESS) {
            pamh = NULL;
            fprintf(stderr, "Failed to release PAM authentication handle\n");
            exit(EXIT_FAILURE);
        }

        // Programın başarılı bir şekilde sonlandırılması
        return (retval == PAM_SUCCESS) ? EXIT_SUCCESS : EXIT_FAILURE;
}
```

Burada **pam_start** fonksiyonu içinde **login** belirttiğimiz için **/etc/pam.d/login** dosyası dikkate alınır.

Önyükleyici

Grub

Grub linux dağıtımlarının genelinde tercih edilen önyükleyicidir.

Kurulum

Grub yüklemek için öncelikle aşağıdaki gibi kurulum yapmalıyız.

```
$ ymp install grub
```

Ardından önyükleyicimizi diske aşağıdaki gibi yüklememiz gerekmektedir.

```
# x86_64-efi UEFI bios için. i386-pc legacy bios için
# /dev/sda kurulacak diskin adı
$ grub-install --target=x86_64-efi /dev/sda
```

Yapılandırma

Grub yapılandırma dosyası **/boot/grub/grub.cfg** dosyasıdır. Bu dosyayı elle yazabilir veya **grub-mkconfig** komutu ile otomatik oluşturabilirsiniz.

```
$ grub-mkconfig -o /boot/grub/grub.cfg
```

Yapılandırma dosyasına yazacağımız komutlar grub konsolunda çalıştırılır. Örneğin dağıtımımızı otomatik olarak başlatmak için:

```
insmod all_video
set root=(hd0,gpt2)
linux /boot/linux-6.13 root=/dev/sda2 rw quiet
initrd /boot/initrd.img-6.13
boot
```

- İlk komut grub ekranında **all_video** modülü yüklemek için
- İkinci komut Hangi diskten açılacağını belirlemek için
- Üçüncü komut çekirdeği ve parametrelerini belirlemek için
- Döndüncü komut initramfs dosyası belirlemek için
- Son komut başlatmak için

Menü oluşturmak için ise aşağıdaki gibi bir yol izlenebilir:

```
menuentry "Linux" --class linux {
   ...
}
```

Burada dikkat edilmesi gereken menü altında boot komutuna ihtiyacımızın olmamasıdır.

Grub konsolu

Eğer bir yapılandırma dosyanız yoksa, dosyada bir hata varsa, veya **c** tuşuna bastıysanız grub konsolu açılır. Bu ekranda elle açılışı sağlayabilirsiniz.

Bu konsolda kullanacağınız komutların listesine **help** ile ulaşabilirsiniz.

```
$ help
```

Var olan diskleri listelemek için **ls** kullanılır. Eğer bir diskin içerisini listelemek isterseniz ise bu komuttan sonra o diski parametre olarak vermelisiniz.

```
$ ls (hd0,gpt2)/
```

set komutu bir değişken ayarlamayı sağlar. **root** değişkeni açılışın yapılacağı diski belirlemenizi sağlar.

```
$ set root=(hd0,gpt2)
```

insmod modül yüklemeyi sağlar. Örneğin:

```
$ insmod ext2
```

linux komutu çekirdeği yüklemeyi ve parametreleri ayarlamayı sağlar

```
$ linux /boot/linux-6.13 rw quiet root=/dev/sda2
```

initrd komutu initramfs dosyasını belirlemeye yarar.

```
$ initrd /boot/initrd.img-6.13
```

exit bir sonraki önyükleyiciye geçmeyi sağlar. Eğer bir sonraki önyükleyici mevcut değilse yeniden başlatır.

```
$ exit
```

terminal_output komutu ile terminal çıktı modunu değiştirebilirsiniz.

```
$ terminal_output console
```

Diskler ve Dosya sistemleri

Dosya Sistemleri

ext4

Ext4 standart dosya sistemdir ve çoğu linux dağıtımında varsayılan olarak tercih edilir.

Diski bağlama

ext4 diskleri bağlamak için öncelikle ext4 çekirdek modülünün etkin olması gerekmektedir. **/sys/fs/ext4** dizini olup olmadığına bakaramk etkın olup olmadığını öğrenebilirsiniz. Eğer bu dizin yoksa modülü yüklemek için aşağıdaki komutu kullanabilirsiniz.

```
$ modprobe ext4
```

Daha sonra aşağıdakı komut yardımı ile diski bağlayabiliriz.

```
# diski yazılabılır şekilde bağlamak için rw salt okunur bağlamak için ro kullanılır.
$ mount -t ext4 -o defaults,rw /dev/sda1 /baglama/noktasi
```

Ext4 biçimlendirme

mkfs.ext4 komutu e2fsprogs paketi ile sağlanır. Öncelikle e2fsprogs yükleyelim.

```
$ ymp install e2fsprogs
```

Daha donra diski biçimlendirelim.

```
$ mkfs.ext4 /dev/sda1
```

Not: diski biçimlendirmek verilerinize kalıca hasar verebilir.

Günlüklemeyi kapatma

Ext4 dosya istemi günlükleme özelliğine sahiptir. Bu özelliği aşağıdaki gibi kapatabilirsiniz.

```
$ tune2fs -O ^has_journal /dev/sda1
```

Eğer disk biçimlendirirken kapatmak isterseniz aşağıdaki gibi komut kullanabilirsiniz.

```
$ mkfs.ext4 -O ^has_journal /dev/sda1
```

Btrfs

Btrfs, Linux çekirdeği için geliştirilmiş bir dosya sistemidir. Gelişmiş özellikleri ve esnek yapısıyla, modern depolama ihtiyaçlarını karşılamak üzere tasarlanmıştır.

Disk bağlama

btrfs diskleri bağlamak için öncelikle çekirdek modülünün bulunması gerekmektedir. Bunun için **/sys/fs/btrfs** dizini bulunuyormu diye kontrol edebilirsiniz. Eğer bulunmuyorsa aşağıdaki komut ile etkinleştirebilirsiniz.

```
$ modprobe btrfs
```

Ardından diski bağlamak için aşağıdaki komutu kullanabilirsiniz.

```
$ mount -t btrfs -o defaults,rf /dev/sda1 /baglama/noktasi
```

Btrfs biçimlendirme

mkfs.btrfs komutu btrfs-progs paketi ile sağlanır. Öncelikle btrfs-progs yüklemeliyiz.

```
$ ymp install btrfs-progs
```

Ardından diski biçimlendirelim.

```
$ mkfs.btrfs /dev/sda1
```

Not: diski biçimlendirmek verilerinize kalıca hasar verebilir.

Rootfs olarak kullanma

Btrfs dosya sistemini kök dizin olarak kullanmak için çekirdek parametrelerine **rootfstype=btrfs** eklenmesi gerekmektedir. Eğer **grub** kullanıyorsanız **/etc/default/grub** dosyanıza aşağıdaki gibi ekleme yapmalısınız ve yapılandırmanızı güncellemelisiniz.

```
GRUB_CMDLINE_LINUX_DEFAULT="quiet rootfstype=btrfs"
```

Ardından /etc/fstab içerisine aşağıdaki gibi girdi yazmalısınız.

```
UUID=abcdef-0055-4958-990f-1413ed1186ec    /    btrfs    defaults,rw    0    0
```

Subvolume kavramı

Btrfs dosya sisteminde "subvolume", dosya sisteminin içinde bağımsız bir dosya sistemi gibi davranabilen bir alt bölüm veya mantıksal birimdir. Subvolumelar, Btrfs'nin sunduğu esnekliği ve yönetim kolaylığını artıran önemli bir özelliktir.

Bağlamak

Bir btrfs dosya sisteminde subvolume bağlamak için bağlama paraetrelerine **subvol=subvol_adi** şeklinde ekleme yapmalıyız.

```
$ mount -t btrfs -o defaults,rw,subvol=@/yedek /dev/sda1 /yedek
```

Bununla birlikte bir subvolume varsayılan ayarlanıp seçenek kullanılmadan da bağlanabilir.

```
# subvol-id değerini subvolume listesinden bulabilirsiniz.
$ btrfs subvol set-default subvol-id /
```

Oluşturma

Bir subvolume oluşturmak için aşağıdaki komutu kullanabilirsiniz

```
# /bagli/dizin/ dizinine btrfs dosya sistemi bağlı olduğunu düşünebilirsiniz.
$ btrfs subvol create /bagli/dizin/yeni_subvolume
```

Listeleme

Btrfs üstündeki subvolume listesi için aşağıdaki komutu kullanabilirsiniz.

```
# /bagli/dizin/ dizinine btrfs dosya sistemi bağlı olduğunu düşünebilirsiniz.
$ btrfs subvol list -t /bagli/dizin/
```

Silme

Btrfs dosya sisteminde subvolume silmek için:

```
# /bagli/dizin/ dizinine btrfs dosya sistemi bağlı olduğunu düşünebilirsiniz.
$ btrfs subvol delete /bagli/dizin/silinecek_subvolume
```

Yeniden adlandırma

Btrfs dosya sisteminde bir subvolume adını değiştirmek için basitçe **mv** komutunu kullanabilirsiniz.

```
$ mv /bagli/dizin/@dosyalar /bagli/dizin/eski-dosyalar
```

Yedekleme (snapshot)

Btrfs dosya sisteminde bir subvolume yedeği alınabilir. Bunun için aşağıdaki komutu kullanabilirsiniz.

```
$ btrfs subvol snapshot /bagli/dizin/@dosyalar /bagli/dizin/@yedek
```

Bu sayede subvolume yedeğini almış olduk. Yedekten geri dönmek için mevcut subvolume silinip yedeğin yedeği alınabilir.

Zram

Linux sistemlerinde bellek yönetimini optimize etmek ve performansı artırmak için kullanılan bir yöntemdir. Zram, sıkıştırılmış bellek blokları oluşturarak, fiziksel RAM üzerinde daha fazla veri depolamaya olanak tanır.

Yapılandırma

Zram'ı yapılandırmak için, öncelikle zram cihazını oluşturmalısınız. Aşağıdaki adımları izleyerek zram'ı yapılandırabilirsiniz:

1. **Zram Cihazını Oluşturma:** Aşağıdaki komut ile zram cihazını oluşturabilirsiniz. *num_devices* değeri, oluşturmak istediğiniz zram cihazı sayısını belirtir. Örneğin, 2 zram cihazı oluşturmak için:

```
modprobe zram num_devices=2
```

2. **Sıkıştırma Algoritmasını Belirleme:** Zram cihazının sıkıştırma algoritmasını ayarlamak için:

```
echo lz4 > /sys/block/zram0/comp_algorithm
```

3. **Disk Boyutunu Ayarlama:** Zram cihazının disk boyutunu belirlemek için:

```
echo 1G > /sys/block/zram0/disksize
```

Not: *disksize* değerini sisteminizin RAM miktarına göre ayarlayabilirsiniz. Örneğin, 1G yerine 2G veya daha fazla bir değer verebilirsiniz.

4. **Swap Alanı Olarak Kullanma:** Zram cihazını swap alanı olarak kullanmak için:

```
mkswap /dev/zram0
swapon /dev/zram0
```

Kullanım

Zram, genellikle swap alanı olarak kullanılır. Yukarıdaki adımlarla zram'ı etkinleştirdikten sonra, sisteminizin bellek yönetimi otomatik olarak zram'ı kullanmaya başlayacaktır. Bellek kullanımı arttıkça, zram sıkıştırılmış verileri RAM üzerinde depolayarak sistem performansını artırır.

Zram'ı devre dışı bırakmak için aşağıdaki komutları kullanabilirsiniz:

```
swapoff /dev/zram0
echo 0 > /sys/block/zram0/disksize
```

Bu komutlar, zram'ı devre dışı bırakacak ve ilgili kaynakları serbest bırakacaktır.

Zram Durumunu Kontrol Etme

Zram'ın durumunu kontrol etmek için aşağıdaki komutu kullanabilirsiniz:

```
cat /proc/swaps
```

Bu komut, sistemdeki mevcut swap alanlarını ve zram ile ilgili bilgileri gösterecektir.

Cryptsetup

Linux dağıtımlarında genellikle disk şifrelemek için tercih edilen yöntemlerden biridir.

Kurulum

ymp ile yüklemek için aşağıdaki komutu kullanabilirsiniz:

```
ymp install cryptsetup
```

Kullanımı

Öncelikle diski şifrelemek için formatlayalım.

```
cryptsetup luksFormat /dev/sda
```

Not: bu işlem tüm diski silecektir. Veri kaybına sebep olabilir. Yedek almayı unutmayın.

Ardından diski etkinleştirelim.

```
cryptsetup luksOpen /dev/sda example
```

Bu şekilde diski çözdükten sonra **/dev/mapper/example** adında bir block aygıt oluşturulur. Bu block aygıtı bölümlendirip kullanabilirsiniz.

Çözülmüş diskinizi tekrar kapatmak için öncelikle diskin tüm bölümlerinin bağlantısını kesmelisiniz.

```
umount /dev/mapper/example*
```

Ardından diski kapatabiliriz.

```
cryptsetup luksClose example
```

Parola değiştirme

Öncelikle parola eklemiz gerekmektedir. Bunun için aşağıdaki gibi bir yol kullanmalıyız.

```
cryptsetup luksAddKey /dev/sda
```

Daha sonra eski parolamızı kaldıralım.

```
cryptsetup luksRemoveKey /dev/sda
```

Swap (Takas Alanı)

Swap alanı belleğin yeterli olmadığı durumda adreslemek için kullanılan disk alanıdır. Bu sayede RAM tamamen dolmaya başladığında bazı veriler swap alanına taşınarak sistemin çakılması önlenmiş olur.

Not: Diskin okuma/yazma hızı RAM kadar fazla olmadığı için disk üzerinde swap kullanmak performans sorunlarına sebep olabilir.

Swap alanı oluşturulması

Swap oluşturmak için bir dosya kullanabilir veya diskin bir bölümünü swap olarak kullanabilirsiniz.

Disk bölümünden swap oluşturulması

Öncelikle disk bölümümüzü formatlayalım.

```
# Burada swap bölümümüzün sda5 olduğunu varsaydık.
mkswap /dev/sda5
```

Dosyadan swap oluşturulması

Bir dosya oluşturup formatlayalım.

```
# 4Gb swap dosyası oluşturduk.
dd if=/dev/zero of=/swapfile bs=4M count=1000
# gereken iznini ayarladık.
chmod 0600 /swapfile
# formatlayalım
mkswap /swapfile
```

Bu işlem diske 0 yazarak çalışır. Diskin hızına bağlı olarak uzun sürebilir. Bunun yerine şu şekilde de oluşturabiliriz.

```
# dosya açalım
truncate -s 4G /swapfile
# gereken iznini ayarladık.
chmod 0600 /swapfile
# loop haline getirelim.
loop=$(losetup --find --show /swapfile)
# loop algıtını formatlayalım
mkswap "$loop"
```

Swap alanının etkinleştirilmesi

Swap alanını **swapon** komutu kullanarak etkinleştirebiliriz.

```
swapon /dev/sda5
```

Kontrol etmek için **/proc/swaps** dosyasına bakabilirsiniz.

```
cat /proc/swaps
>> Filenam    Type      Size    Used Priority
>> /dev/sda5  partition 4194300 0    -2
```

Swap alanının kapatılması

Swap alanını kapatmak için **swapoff** komutu kullanabilirsiniz.

```
swapoff /dev/sda5
```

Network Block Device (nbd)

Network Block Device (nbd), bir ağ üzerindeki disk görüntülerini veya diğer blok cihazlarını uzak bir sistemde kullanmanızı sağlayan bir sistem aracıdır. NBD, istemci-sunucu mimarisi kullanarak çalışır; istemci, NBD sunucusuna bağlanarak uzak bir blok cihazına erişim sağlar. Bu, veri depolama ve yedekleme çözümleri için oldukça faydalıdır.

Sunucu tarafı

NBD sunucusu oluşturmak için **qemu-nbd** kullanabilirsiniz. İşlem adımları aşağıdaki gibidir:

Disk İmajı Oluşturma

Öncelikle disk imajı oluşturmalısınız:

```
$ qemu-img create -f qcow2 test.img 50G
```

NBD Sunucusunu Başlatma

Ardından, NBD sunucusunu başlatmak için şu komutu kullanabilirsiniz:

```
# Salt okur yapmak için -r parametresi kullanılır.
# -b 0.0.0.0 dışa açılacak adres
# -p port numarası
# -e en fazla bağlantı sayısı
$ qemu-nbd -f qcow2 -b 0.0.0.0 -p 10809 -e 10 test.img
```

Kullanıcı tarafı

Kullanıcı tarafında ise aşağıdaki adımları takip etmelisiniz:

NBD Çekirdek Modülünü Yükleme

Öncelikle nbd çekirdek modülünü yükleyin:

```
# nbds_max block aygıt sayısı
$ modprobe nbd nbds_max=1
```

Bağlantıyı Ayarlama

Bağlantıyı ayarlamak için aşağıdaki komutu kullanın:

```
# Buradaki host ve portu kendinize göre değiştirin.
$ nbd-client 192.168.1.31 /dev/nbd0 -p 10809
```

Bölüm Listesini Güncelleme

Son olarak, varsa bölümlerin listesini güncelleyin:

```
$ partprobe /dev/nbd0
```

Artık **/dev/nbd0** blok aygıtını kullanabilirsiniz.

Fuse

FUSE (Userspace Dosya Sistemi), kullanıcıların özel izinlere ihtiyaç duymadan dosya sistemlerini bağlamalarını sağlayan bir yöntem sunar. Linux'ta genellikle yönetici ayrıcalıklarına sahip olanlar tarafından gerçekleştirilen bağlama işlemi, FUSE ile herhangi bir kullanıcı tarafından gerçekleştirilebilir hale gelir.

Kurulumu

Türkmen linuxta fuse yüklemek için aşağıdaki komutu kullanabilirsiniz.

```
$ ymp install fuse
# Eğer libfuse2 gerektiren bir uygulama çalıştıracaksanız şunu da yüklemelisiniz.
$ ymp install fuse2
```

Daha sonra fuse servisini çalıştıralım.

```
# Açılışa ekleyelim
$ rc-update add fuse
# Çalıştıralım
$ rc-service fuse start
```

Eğer servis yerine elle başlatmak isterseniz aşağıdaki gibi çalıştırabilirsiniz.

```
# Önce modülü yükleyelim.
$ modprobe fuse
# Şimdi connections kısmını bağlayalım.
$ mount -t fusectl none /sys/fs/fuse/connections
```

Son olarak fusermount komutuna suid biti ayarlayalım.

```
$ chmod u+s /usr/bin/fusermount
```

Yapılandırma

FUSE için kullanılabilir yapılandırma dosyaları şunlardır:

```
/etc/fuse.conf
```

fuse.conf dosyasında iki yapılandırma değişkeni bulunur:

- mount_max: Kök olmayan kullanıcılara izin verilen maksimum FUSE bağlantı sayısını ayarlar (varsayılan olarak ayarlanmamışsa 1000'dir).
- user_allow_other: Kök olmayan kullanıcıların allow_other veya allow_root bağlama seçeneklerini belirtmesine izin verir. Bu, güvenlik nedenleriyle devre dışı bırakılmıştır.

Fuse ile dosya sistemi yazma

fuse kullanarak dosya sistemi oluşturabiliriz. Aşağıda örnek bir dosya sistemi kodu bulunmaktadır:

```c
#define FUSE_USE_VERSION 30
#include <fuse.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <fcntl.h>

// Dosya ve dizin özniteliklerini döndüren işlev
static int hello_getattr(const char *path, struct stat *stbuf) {
    int res = 0;

    memset(stbuf, 0, sizeof(struct stat));
    if (strcmp(path, "/") == 0) {
        // Root dizin öznitelikleri
        stbuf->st_mode = S_IFDIR | 0755;
        stbuf->st_nlink = 2;
    } else if (strcmp(path+1, "hello") == 0) {
        // "hello" dosyasının öznitelikleri
        stbuf->st_mode = S_IFREG | 0444;
        stbuf->st_nlink = 1;
        stbuf->st_size = 12; // "Hello World!\n" uzunluğu
    } else {
        // Hata durumu: Dosya veya dizin bulunamadı
        res = -ENOENT;
    }

    return res;
}

// Dizini okuyan işlev
static int hello_readdir(const char *path, void *buf, fuse_fill_dir_t filler,
                         off_t offset, struct fuse_file_info *fi) {
    (void) offset;
    (void) fi;

    if (strcmp(path, "/") != 0)
        return -ENOENT;

    // Root dizini içeriğini doldur
    filler(buf, ".", NULL, 0, 0);
    filler(buf, "..", NULL, 0, 0);
    filler(buf, "hello", NULL, 0, 0);

    return 0;
}

// Dosyayı açan işlev
static int hello_open(const char *path, struct fuse_file_info *fi) {
    if (strcmp(path+1, "hello") != 0)
        return -ENOENT;
```

```
    // Salt okunur olarak dosyayı aç
    if ((fi->flags & 3) != O_RDONLY)
        return -EACCES;

    return 0;
}

// Dosyadan okuma işlevi
static int hello_read(const char *path, char *buf, size_t size, off_t offset,
                      struct fuse_file_info *fi) {
    size_t len;
    (void) fi;
    if(strcmp(path+1, "hello") != 0)
        return -ENOENT;

    // "Hello World!\n" içeriğini dosyadan oku
    char *hello_str = "Hello World!\n";
    len = strlen(hello_str);
    if (offset < len) {
        if (offset + size > len)
            size = len - offset;
        memcpy(buf, hello_str + offset, size);
    } else
        size = 0;

    return size;
}

// FUSE işlevlerini tanımlayan yapı
static struct fuse_operations hello_oper = {
    .getattr = hello_getattr,
    .readdir = hello_readdir,
    .open = hello_open,
    .read = hello_read,
};

// Ana fonksiyon
int main(int argc, char *argv[])
{
    return fuse_main(argc, argv, &hello_oper, NULL);
}
```

Kodu derlemek için aşağıdaki komut kullanılır.

```
gcc -o hello hello.c `pkg-config --cflags --libs fuse3`
```

Bu basit örnek, FUSE kullanarak "hello" adında bir dosya oluşturur ve bu dosyanın içeriği "Hello World!" cümlesiyle doldurur. Dosya sistemine "/hello" yolunu kullanarak erişilebilir. Bu kod, temel dosya sistemlerinin işlevlerini (getattr, readdir, open, read) uygular. Örneğin, dosyanın özniteliklerini alma (getattr), dizin içeriğini okuma (readdir), dosyayı açma (open) ve dosyadan okuma (read) işlemleri gerçekleştirilir.

Disk Yönetimi

Udisks

Udisks disk bağlama ve düzenleme özelliklerini sağlayan dbus arayüzü ve **udisksctl** komutunu sağlar. Grafik arayüzlü dosya yöneticileri udisks kullanarak disk bağlama işlemlerini gerçekleştirir.

Kurulum

Udisks çalışması için polkit ve dbus gerekmektedir. ymp ile yüklemek için:

```
$ ymp install udisks2
```

Servisin başlatılması

udisk servis yardımı ile çalışır.

Openrc servisin başlatmak için:

```
# Açılışa eklemek için
$ rc-update add udisks2 default
# Başlatmak için
$ rc-service udisks2 start
```

Kullanım

Udisks servisine udisksctl komutu ile komut yollanır.

```
Usage:
  udisksctl COMMAND

Commands:
  help            Shows this information
  info            Shows information about an object
  dump            Shows information about all objects
  status          Shows high-level status
  monitor         Monitor changes to objects
  mount           Mount a filesystem
  unmount         Unmount a filesystem
  unlock          Unlock an encrypted device
  lock            Lock an encrypted device
  loop-setup      Set-up a loop device
  loop-delete     Delete a loop device
  power-off       Safely power off a drive
  smart-simulate  Set SMART data for a drive
```

Bir disk bağlamak için aşağıdakine benzer komut kullanılır.

```
$ udisksctl mount -b /dev/sdc1
```

Bağı sökmek için:

```
$ udisksctl unmount -b /dev/sdc1
```

Yapılandırma

Yapılandırma dosyasına **/etc/udisks2/udisks2.conf** içinden ulaşabilirsiniz. Örneğin fat32 ve exfat dosya sistemlerinin fmask ve dmask değerlerini ayarlamak için:

```
[defaults]
# tüm dosyaların ve dizinlerin izinlerini 755 olarak görmek için
vfat_defaults=fmask=022,dmask=022,uid=$UID,gid=$GID
exfat_defaults=fmask=022,dmask=022,uid=$UID,gid=$GID
```

Yetlkilendirme

Disk bağlayan polkit kuralının adı **org.freedesktop.udisks2.filesystem-mount** olmaktadır.

Örneğin parolasız disk bağlamak için aşağıdaki gibi polkit kuralı yazabilirsiniz:

```
polkit.addRule(function(action, subject){
    if (action.id == "org.freedesktop.udisks2.filesystem-mount"){
        return polkit.Result.YES;
    }
});
```

Not: Türkmen linux varsayılan olarak udisks parolasız bağlamaya izin vermez. Bunun kaldırmak için **/usr/share/polkit-1/rules.d/49-auth-udisks2.rules** dosyasını silebilirsiniz.

Kurulum

Basit Kurulum

Bu bölümde **Ext4** dosya sistemine grub kullanarak kurulum anlatılacaktır. Anlatım boyunca **/dev/sda** diski üzerinden örnekleme yapılmıştır. Siz kendi diskinize göre düzenleyebilirsiniz.

Uefi - Legacy tespiti

/sys/firmware/efi dizini varsa uefi yoksa legacy sisteme sahipsinizdir. Eğer uefi ise ia32 veya x86_64 olup olmadığını anlamak için **/sys/firmware/efi/fw_platform_size** içeriğine bakın.

```
[[ -d /sys/firmware/efi/ ]] && echo UEFI || echo Legacy
[[ "64" == $(cat/sys/firmware/efi/fw_platform_size) ]] && echo x86_64 || echo ia32
```

Disk Bölümlendirme

Uefi kullananlar ayrı bir disk bölümüne ihtiyaç duyarlar. Bu bölümü **fat32** olarak bölümlendirmeliler.

Bu anlatımda kurulum için **/boot** dizinini ayırmayı ve efi bölümü olarak aynı diski kullanmayı tercih edeceğiz.

Öncelikle **cfdisk** veya **fdisk** komutları ile diski bölümlendirelim.

```
$ cfdisk /dev/sda
```

Ardından boot bölümünü ve kök dizini formatlayalım.

```
$ mkfs.vfat /dev/sda1
$ mkfs.ext4 /dev/sda2
```

Not: ext4 dosya sistemi araçları **e2fsprogs** ile sağlanır.

Eğer /boot bölümünü ayırmayacaksanız grub yüklenirken **unknown filesystem** hatası almanız durumunda aşağıdaki yöntemi kullanabilirsiniz.

```
$ e2fsck -f /dev/sda2
$ tune2fs -O ^metadata_csum /dev/sda2
```

Dosya sistemini kopyalama

Kurulacak sistemin imajını bir dizine bağlayalım.

```
# /dev/loop0 bağlayalım
$ mount -o loop /dev/loop0 /source
```

Şimdi de bölümlerimizi bağlayalım.

```
# /target yoksa oluşturun.
$ mount -t ext4 /dev/sda2 /target
$ mkdir -p /target/boot
$ mount -t vfat /dev/sda1 /target/boot
```

Ardından dosyaları kopyalayalım.

```
# -p dosya izinlerini korur
# -r alt dizinlerle beraber kopyalar
# -f soru sormayı kapatır
# -v detaylı çıktıları gösterir
$ cp -prfv /source/* /target
```

Daha sonra diski senkronize edelim.

```
$ sync
```

Bootloader kurulumu

Sisteme **ymp chroot** komutu ile girelim.

```
$ ymp chroot /target
# Bunun yerine aşağıdaki gibi de girilebilir.
for dir in /dev /sys /proc /run /tmp ; do
        mount -bind /$dir /target/$dir
done
$ chroot /target
```

Şimdi de eğer uefi kullanıyorsanız efivar bağlayalım.

```
$ mount -t efivarfs efivarfs /sys/firmware/efi/efivars
```

Grub paketini yükleyelim.

```
$ ymp install grub
```

Son olarak grub kurulumu yapalım.

```
# biz /boot ayırdığımız ve efi bölümü olarak kullanacağız.
# uefi kullanmayanlar --efi-directory belirtmemeliler.
# kurulu sistemden bağımsız çalışması için --removable kullanılır.
$ grub-install --removable --boot-directory=/boot --efi-directory=/boot /dev/sda
```

Grub yapılandırması

Öncelikle uuid değerimizi bulalım.

```
$ blkid | grep /dev/sda2
/dev/sda2: UUID="..." BLOCK_SIZE="4096" TYPE="ext4" PARTUUID="..."
```

Şimdi aşağıdaki gibi bir yapılandırma dosyası yazalım ve /boot/grub/grub.cfg dosyasına kaydedelim. Burada uuid değerini ve çekirdek sürümünü düzenleyin.

```
search --fs-uuid --no-flopy --set=root <uuid-değeri>
linux /boot/vmlinuz-<çekirdek-sürümü>   root=UUID=<uuid-değeri> rw quiet
initrd /boot/initrd.img-<çekirdek-sürümü>
boot
```

Ayrıca otomatik yapılandırma da oluşturabiliriz.

```
$ grub-mkconfig -o /boot/grub/grub.cfg
```

Fstab dosyası

Bu dosyayı doldurarak açılışta hangi disklerin bağlanacağını ayarlamalıyız. **/etc/fstab** dosyasını aşağıdakine uygun olarak doldurun.

```
# <fs>                  <mountpoint>    <type>          <opts>          <dump/pass>
/dev/sda1       /boot   vfat    defaults,rw     0       1
/dev/sda2       /       ext4    defaults,rw     0       1
```

Not: Disk bölümü konumu yerine **UUID="<uuid-değeri>"** şeklinde yazmanızı öneririm. Bölüm adları değişebilirken uuid değerleri değişmez.

Yapılandırma

Dil ayarlama

Sistem dilini ayarlamak için öncelikle **/etc/locale.gen** dosyamızı aşağıdaki gibi düzenleyelim.

- Dil kodlarına **/usr/share/i18n/locales** içerisinden ulaşabilirsiniz.
- Karakter kodlamalara **/usr/share/i18n/charmaps** içinden ulaşabilirsiniz.

```
tr_TR.UTF-8 UTF-8
```

Not: En altta boş bir satır bulunmalıdır.

Ardından /lib64/locale dizini yoksa oluşturalım.

```
mkdir -p /lib64/locale/
```

Şimdi de çevresel değişkenlerimizi ayarlamak için **/etc/profile.d/locale.sh** dosyamızı düzenleyelim.

```
#!/bin/sh
# Language settings
export LANG="tr_TR.UTF-8"
export LC_ALL="tr_TR.UTF-8"
```

Not: Türkçe büyük küçük harf dönüşümü (i -> İ ve ı -> I) ascii standartına uyumsuz olduğu için **LC_ALL** kısmını türkçe ayarlamayı önermiyoruz. Bunun yerine **C.UTF-8** veya **en_US.UTF-8** olarak ayarlayabilirsiniz.

Son olarak **locale-gen** komutunu çalıştıralım.

```
locale-gen
```

Eğer **/lib64/locale/** dizine okuma iznimiz yoksa verelim.

```
chmod 755 -R /lib64/locale/
```

Son olarak sistemi yeniden başlatalım.

```
openrc-shutdown -r
```

Hostname ayarlama

Makina adımızı ayarlamak için öncelikle belirlediğimiz makina adını **/etc/hostname** dosyasına yazalım.

```
$ echo "sunucu01" > /etc/hostname
```

Ardından hostname servisini yeniden başlatalım.

```
$ rc-service hostname restart
```

Eğer makina adının her açılışta aynı kalmasını istiyorsak servisi etkinleştirelim.

```
$ rc-update add hostname
```

Servis yöneticisini kullanmadan **hostname** komutunu kullanarak makina adını değiştirmek mümkündür.

```
$ hostname sunucu01
```

Not: Makina adı belirlerken belirlerken aşağıdaki kurallara dikkat etmelisiniz:

- büyük harf içermemeli
- ilk harf sayı olmamalı
- 253 karakterden uzun olmamalı
- türkçe karakter içermemeli

Zaman dilimi ayarlama

Öncelikle saat ayarlamak için **tzdata** paketine ihtiyacımız bulunmaktadır. Bu paket saat dilimlerini ayarlamaya yarar.

ymp ile yüklemek için:

```
$ ymp install tzdata
```

Saat dilimi ayarımız **/etc/localtime** içerisinde bulunur. Örneğin **UTC** için

```
$ cat /etc/localtime
<-00>0
```

Saati şehrinize göre ayarlamak için **/usr/share/zoneinfo/** içerisinden uygun olanı bulup buraya sembolik bağlama yapalım.

```
$ ln -s /usr/share/zoneinfo/Asia/Istanbul /etc/localtime
```

Eğer saatiniz hatalı ise ntpd kullanarak internet üzerinden güncellemesini sağlayabilirsiniz.

Araçlar

syslog

Syslog çalışan sistem servislerinin loglarını tutar. Bu loglar /var/log/messages içerisinde bulunur. Bu sayede sistemle ilgili tüm logları buradan takip edebilirsiniz.

```
# logları takip etmek için aşağıdaki komut kullanılabilir.
$ tail -f /var/log/messages
```

Servisin başlatılması

syslogd komutunu kullanarak servisi başlatabilirsiniz. Bu komut busybox tarafından sağlanır.

```
# komutla ilgili detaylara --help parametresi ile ulaşabilirsiniz.
$ syslogd
```

Eğer isterseniz aşağıdaki gibi bir openrc servisi yazıp kullanabilirsiniz.

```
#!/sbin/openrc-run

description="Message logging system"

name="busybox syslog"
command="/bin/busybox"
command_args="syslogd -n"
pidfile="/run/syslogd.pid"
command_background=true

depend() {
      need clock hostname localmount
      provide logger
}
```

Log yazma

logger komutunu kullanarak log yazdırabilirsiniz.

```
$ logger "hello world"
```

bununla birlikte aşağıdaki C kodunu kullanarak log yazmak mümkündür.

```c
#include <syslog.h>
#include <unistd.h>
int main(int argc, char** argv){
    setlogmask (LOG_UPTO (LOG_NOTICE));
    openlog ("test", LOG_CONS | LOG_PID | LOG_NDELAY, LOG_LOCAL1);
    syslog (LOG_NOTICE, "Hello World from %d", getuid ());
    closelog ();
    return 0;
}
```

Not: **/dev/log** konumundaki unix sockete bağlanıp log yollamayı oradan da yapabilirsiniz.

ntpd

Sistem saatini bir sunucudan bakarak eşlemeye yarayan araçtır. **busybox** tarafından sağlanabilir.

```
BusyBox v1.36.1 (2023-06-09 19:35:59 UTC) multi-call binary.

Usage: ntpd [-dnqNwl] [-I IFACE] [-S PROG] [-k KEYFILE] [-p [keyno:N:]PEER]...

NTP client/server

        -d[d]   Verbose
        -n      Run in foreground
        -q      Quit after clock is set
        -N      Run at high priority
        -w      Do not set time (only query peers), implies -n
        -S PROG Run PROG after stepping time, stratum change, and every 11 min
        -k FILE Key file (ntp.keys compatible)
        -p [keyno:NUM:]PEER
                Obtain time from PEER (may be repeated)
                Use key NUM for authentication
                If -p is not given, 'server HOST' lines
                from /etc/ntp.conf are used
        -l      Also run as server on port 123
        -I IFACE Bind server to IFACE, implies -l
```

Servisin başlatılması

Servisi elle başlatmak için *busybox ntpd* komutunu kullanabiliriz.

```
$ busybox ntpd
```

Openrc servisi kullanarak da başlatabiliriz. Bunun için ağağıdaki gibi servis dosyası yazalım.

```
#!/sbin/openrc-run

description="Network Time Sync"

name="busybox ntpd"
command="/bin/busybox"
command_args="ntpd -n"
pidfile="/run/syslogd.pid"
command_background=true

depend() {
      need net
      provide timesync
}
```

Bu servis dosyasını **/etc/init.d/ntpd** dosyasına kaydettikten sonra aşağıdaki komutla etkinleştirelim.

```
# etkinleştirelim
$ rc-update add ntpd
# başlatalım
$ rc-service ntpd start
```

Yapılandırma dosyası

Yapılandırma dosyası **/etc/ntp.conf** konumundadır. Örnek yapılandırma aşağıdaki gibidir.

```
server 0.pool.ntp.org
server 1.pool.ntp.org
server 2.pool.ntp.org
```

USB/IP

USB/IP protokolü, USB cihazlarının ağ üzerinden paylaşımına olanak tanır. Bu sayede usb aygıtlarını ağ üzerinden başka bir cihaz ile kullanmak mümkün hale gelir.

Kurulum

usbip komutu linux çekirdeğinin kod deposu ile beraber gelmektedir.

Türkmen linux üzerine kurmak için:

```
$ ymp install linux-tools
```

Kurulumu yaptıktan sonra çekirdek modülünün açılışta yüklenmesi için:

```
$ echo "usbip_host" > /etc/modules-load.d/usbip.conf
$ echo "vhci_hcd" >> /etc/modules-load.d/usbip.conf
$ modprobe usbip_host
$ modprobe vhci_hcd
```

Sunucu tarafı

Öncelikle servisi başlatalım.

```
$ usbipd --ipv4 --debug
```

Not: Varsayılan 3240 portu üzerinde ağ üzerinde paylaşılmaktadır. Herhangi bir doğrulama mekanizması olmadığı için güvenlik açısından sorun oluşturabilir.

Ardından usb listesini bulalım.

```
$ usbip list -l
```

Son olarak usb paylaşıma açalım:

```
$ usbip bind -b 3-1.5
```

Paylaşımı sonlandırmak için:

```
$ usbip unbind -b 3-1.5
```

Kullanıcı tarafı

Öncelikle sunucu tarafındani usb listesini alalım.

```
$ usbip list -r 10.0.3.1
```

Daha sonra usb bağlantısını sağlayalım.

```
$ usbip attach -r 10.0.3.1 3-1.5
```

Bağlantıyı sonlandırmak için:

```
$ usbip detach -r 10.0.3.1 3-1.5
```

Container

Docker

Docker Kurulumu

Ymp kullanarak docker yüklemek için aşağıdaki komutu kullanabilirsiniz.

```
$ ymp install docker
```

Kaynak koddan yüklemek için sırası ile şunları derleyin

```
https://github.com/krallin/tini
https://github.com/opencontainers/runc
https://github.com/containerd/containerd
https://github.com/moby/moby
https://github.com/docker/cli/
```

Daha sonra container içerisinde ağ bağlantısını yönlendirebilmemiz için aşağıdaki eklemeyi yapmamız gereklidir.

```
# sysctl.conf dosyasına ekleme yapalım
echo "net.ipv4.ip_forward = 1" > /etc/sysctl.conf
sysctl -p /etc/sysctl.conf
# Veya kernel üzerinden etkinleştirelim (Bunu her açılışta yapmamız gerekir)
echo 1 > /proc/sys/net/ipv4/ip_forward
```

Servisin başlatılması

Openrc servisini başlatmak için aşağıdaki komutu kullanabiliriz.

```
# servisi aşılışa eklemek için
$ rc-update add docker
# servisi çalıştırmak için
$ rc-service docker start
```

Servisi elle başlatmak için ise aşağıdaki gibi bir yol izlenebilir.

```
$ containerd &
$ dockerd &
```

Cgroup controller değiştirme

Docker cgroup controller olarak cgroupfs veya systemd kullanmaktadır. Eğer cgroup controller olarak yanış ayarlandıysa docker çalışamaz. Bunu elle ayarlamak için docker servisi komutuna aşağıdaki parametre eklenmelidir.

```
$ dockerd --exec-opt native.cgroupdriver=cgroupfs
```

Bu parametreyi **/etc/conf.d/docker** içerisinden ekleyebilirsiniz.

Basit kullanım

docker run komutu, Docker'da yeni bir container başlatmak ve çalıştırmak için kullanılır. Bu komut, bir Docker imajını temel alarak bir container başlatır ve bu containeri belirli ayarlarla çalıştırmanızı sağlar.

İşte docker run komutunun temel kullanımı ve bazı yaygın seçenekler:

```
docker run [OPTIONS] IMAGE [COMMAND] [ARG...]

    OPTIONS: Docker containerinin çalıştırılması sırasında kullanılan seçenekleri belirtir.
    IMAGE: Başlatılacak containerin temel alınacak Docker imajının adını veya ID'sini belirtir.
    COMMAND: containerin içinde çalıştırılacak komutu belirtir (isteğe bağlı).
    ARG: Komutun argümanlarını belirtir (isteğe bağlı).
```

Örnekler:

Basit bir imajı çalıştırma:

```
$ docker run debian
```

Bu komut, resmi debian imajını kullanarak yeni bir container başlatır.

Bağlantılı bir portu ayarlama:

```
$ docker run -p 8080:80 nginx
```

Bu komut, Docker host'unun 8080 portunu, containerin 80 portuna yönlendirir.

Arkaplanda çalıştırma ve isimlendirme:

```
$ docker run -d --name my-container nginx
```

Bu komut, my-container adında bir containeri arkaplanda çalıştırır.

containeri belirli bir komutla çalıştırma:

```
$ docker run debian ls -l
```

Bu komut, debian imajını temel alarak ls -l komutunu çalıştırır ve ardından containeri kapatır.

Ana bilgisayarın dosya sistemini bir containerle paylaşma:

```
$ docker run -v /host/path:/container/path debian
```

Bu komut, ana bilgisayarın belirli bir dizinini (/host/path) containerin belirli bir diziniyle (/container/path) paylaşır.

Çevre değişkenleri tanımlama:

```
$ docker run -e MY_VARIABLE=my_value debian
```

Bu komut, MY_VARIABLE adında bir çevre değişkenini my_value değeriyle tanımlar.

İçindeki bir bağlamı kullanarak çalıştırma:

```
$ docker run -it debian /bin/bash
```

Bu komut, interaktif modda (-it) debian imajını başlatır ve /bin/bash komutunu çalıştırarak kullanıcıyı containerin içine sokar.

Çalışan containerleri görmek

docker ps komutu, çalışan Docker containerlerini listelemek için kullanılır. Bu komut, çalışan containerlerin temel bilgilerini, ID'lerini, isimlerini, başlatıldığı zamanı ve hangi portların bağlandığını gösterir.

İşte docker ps komutunun temel kullanımı ve bazı yaygın seçenekleri:

```
$ docker ps [OPTIONS]
# OPTIONS: Docker containerlerini listelerken kullanılacak seçenekleri belirtir.
```

Örnekler:

Tüm çalışan containerleri listeleme:

```
$ docker ps
```

Bu komut, şu anda çalışan tüm Docker containerlerini listeler.

Tüm containerleri (çalışan ve durmuş) listeleme:

```
$ docker ps -a
```

Bu komut, şu anda çalışan ve durmuş olan tüm Docker containerlerini listeler.

Açık olan containeri kapatmak ve silmek

Açık olan bir Docker containerini kapatmak için **docker stop** komutunu kullanabilirsiniz. İşte basit bir kullanım örneği:

```
$ docker stop CONTAINER_ID
```

Burada **CONTAINER_ID**, durdurmak istediğiniz Docker containerinin kimliğidir. Alternatif olarak, Docker containerine bir isim verdiyseniz, ismi kullanabilirsiniz.

Örneğin:

```
$ docker stop my-container
```

Eğer çalışan tüm Docker containerlerini durdurmak istiyorsanız, aşağıdaki komutu kullanabilirsiniz:

```
$ docker stop $(docker ps -q)
```

Bu komut, **docker ps -q** komutu ile tüm çalışan containerlerin ID'lerini alır ve ardından bu ID'leri kullanarak **docker stop** komutunu uygular.

Containeri durdurduktan sonra, container hala sistemde bulunur ancak çalışmaz durumda olur. Containeri tamamen kaldırmak için **docker rm** komutunu kullanabilirsiniz. Örneğin:

```
$ docker rm CONTAINER_ID
# veya
$ docker rm my-container
```

Eğer çalışan tüm containerleri durdurup ardından kaldırmak istiyorsanız, aşağıdaki komutu kullanabilirsiniz:

```
docker rm $(docker ps -a -q)
```

Bu komut, tüm containerlerin kimliklerini alır ve ardından bu kimlikleri kullanarak **docker rm** komutunu uygular.

Durdurulmuş bir containeri başlatmak ve içine girmek

docker start ve **docker attach** komutları, Docker containerleri üzerinde çalışma ve etkileşim kurma işlemleri için kullanılır.

docker start komutu, durmuş olan bir Docker containerini başlatmak için kullanılır.

İşte temel kullanımı:

```
$ docker start CONTAINER_ID
# veya
$ docker start CONTAINER_NAME
# CONTAINER_ID veya CONTAINER_NAME, başlatmak istediğiniz Docker containerinin ID'si veya adıdır.
```

Örnek:

```
docker start my-container
```

Bu komut, **my-container** adlı bir Docker containerini başlatır. Başlatılan bir container, **docker ps** komutu ile görülebilir.

docker attach komutu, çalışan bir Docker containerine bağlanmak için kullanılır. Bu komut, containerin ana sürecine doğrudan bir terminal bağlantısı sağlar. İşte temel kullanımı:

```
$ docker attach CONTAINER_ID
# veya
$ docker attach CONTAINER_NAME
# CONTAINER_ID veya CONTAINER_NAME, bağlanmak istediğiniz Docker containerinin ID'si veya adıdır.
```

Örnek:

```
$ docker attach my-container
```

Bu komut, my-container adlı bir Docker containerine bağlanır. Bu komutu kullanarak, containerde çalışan bir sürecin çıktısını görebilir ve klavyeden giriş yapabilirsiniz.

Cgroup

Cgroup çalışan süreçleri guruplamak ve kontrol etmek için kullanılır. Bu sayede belli işlemleri kısıtlayabilir veya toplu şekilde yöneteilirsiniz.

Cgroup linux çekirdeğinin bir özelliğidir. Bir çok iş için kullanılabilir. Örneğin servis yöneticisi servisleri çalıştırıp daha sonra sonlandırmak için cgroup kullanır.

Cgroup oluşturma

Öncelikle cgroup dizininin bağlandığından emin olun.

```
mount | grep /sys/fs/cgroup type
>> none on /sys/fs/cgroup type cgroup2 (rw,nosuid,nodev,noexec,relatime,nsdelegate)
```

Eğer bağlı değilse aşağıdaki gibi bağlayabilirsiniz:

```
mount -t cgroup2 none /sys/fs/cgroup
```

Bir cgroup oluştumak için bu dizine bir alt dizin oluşturmalıyız. Çekirdeğimiz bizim için cgroup oluşturacakdır.

```
mkdir /sys/fs/group/deneme
```

Süreç ekleme

Bir gurubun içerisine aşyağıdaki gibi bir süreç ekleyebilirsiniz.

```
# örneğin pid değerimiz 1234 olsun
echo 1234 > /sys/fs/cgroup/deneme/cgroup.procs
```

Not: eklediğimiz sürecin tüm alt süreçleri de bu guruba dahil olacakdır.

Aynı şekilde cgroup içindeki süreçlerin pid değerlerinin listesini öğrenebiliriz.

```
cat /sys/fs/cgroup/deneme/cgroup.procs
>> 1234
```

Cgroup silme

Öncelikle cgroup içerisindeki tüm süreçleri kapatalım.

```
echo 1 > /sys/fs/cgroup/deneme/cgroup.kill
```

Ardından dizinimizi silelim

```
rmdir /sys/fs/cgroup/deneme/
```

netns (Network Namespace)

netns (network namespace), ağ kaynaklarını izole etmek için kullanılan bir mekanizmadır. Birden fazla ağ yığınını tek bir Linux çekirdek üzerinde çalıştırarak, her bir yığın için ayrı ağ ayarları, araçları ve protokolleri oluşturma imkanı sunar.

netns oluşturma ve kaldırma

Bir netns oluşturmak için aşağıdaki komut kullanılır:

```
$ ip netns add deneme
```

Silmek için ise şu komut kullanılır:

```
$ ip nets del deneme
```

Var olanları listelemek için:

```
$ ip nets list
```

netns içerisinde komut çalıştırma

Bir netns içerisinde komut çalıştırarak o komutun izole bir ağ ortamında çalışması sağlanabilir. Bunun için şu komuttan yararlanılabilir.

```
$ ip nets exec deneme /bin/bash
```

Bu komut hiçbir ağ arayüzü eklemediğimiz için ağ bağlantısı bulunmayan *deneme* ağında çalıştırılır.

netns içine ağ arayüzü eklenmesi

Bir ağ arayüzünü netns içine ekleyebilirsiniz. birden çok ethernet ile çalışmanız gerektiğinde kullanışlı olabilmektedir.

```
# eth1 arayüzünü eklemek için (2. ethernet arayüzü)
$ ip link set eth1 netns deneme
# loop arayüzünü açalım (127.0.0.1 için)
$ ip netns exec deneme ip link set up lo
# dhcp ile ip almak için
$ busybox udhcpc -i eth1
# netns içinde firefox başlatmak için (pingu kullanıcısı ile)
$ ip netns exec deneme su pingu -c "firefox"
```

netns arası iletişim sağlanması

Öncelikle **veth** arayüzü eklememiz gerekmektedir.

```
$ ip link add veth0 type veth peer name veth1
```

Burada veth0 ve veth1 adında 2 adet veth arayüzümüz oluşmaktadır. Arayüzlernden birini netns içine alalım.

```
$ ip link set veth1 netns deneme
```

Ardından arayüzlere birer ip adresi atayalım.

```
# açık hale getirelim
$ ip link set up veth0
$ ip netns exec deneme ip link set up veth1
# loop arayüzünü açalım (127.0.0.1 için)
$ ip netns exec deneme ip link set up lo
# ip atayalım
$ ip addr add 10.0.3.1/24 dev veth0
$ ip netns exec deneme ip addr add 10.0.3.2/24 dev veth1
# varsayılan route verelim
$ ip netns exec deneme ip route add default via 10.0.3.1 dev veth1
# yetkilendirme verelim
$ iptables -t nat -A POSTROUTING -o veth0 -j MASQUERADE
$ iptables -A FORWARD -i veth0 -j ACCEPT
$ iptables -A FORWARD -o veth0 -j ACCEPT
```

Artık deneme adındaki netns içerisinden ana makinamıza **10.0.3.1** adresi ile ulaşabilirsiniz. Ana makinanızdan ise **10.0.3.2** adresi ile netns içerisine ulaşabilirsiniz.

```
# netns içinde 8000 portunda tcp dinlemek için
$ ip netns exec deneme netcat -l -p 8000
# netns içindeki 8000 portuna bağlanmak için
$ netcat 10.0.3.2 8000
```

Dış ağa bağlantı sağlanması

Dış ağa bağlantı sağlanabilmesi için yönlendirmenin açık olması ve ağ kuralı gerekmektedir. Dışa erişim ayarlayalım.

```
$ sysctl net.ipv4.ip_forward=1
$ iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

Artık **eth0** üzerinden dış ağa erişim sağlanabilir.

Geliştirme ortamı

Python

Python kurulumu

Python yüklemek için **ymp install python** komutunu kullanmalısınız.

```
$ ymp install python
```

Kaynak koddan yüklemek için aşağıdaki adımları takip ediniz.

1. https://python.org adresinden kaynak kodu indirin ve bir dizine açın.
2. kaynak kodun içerisinde aşağıdaki komutları kullanarak derleyin.

```
$ autoreconf -fvi
$ ./configure
$ make
$ make install
```

Türkmen linux size birden çok python sürümünü aynı anda kullanmasına olanak tanır. Bunun için **pydefault** komutu kullanarak varsayılan sürümü değiştirebilirsiniz.

```
$ pydefault 3.10
```

Not: python paketi kurulurken var olan en üst sürümü varsayılan olarak ayarlamaktadır.

Not: En üst sürümü kullanmamak sistemin ve uygulamaların düzgün çalışmamasına sebep olabilir.

Pip etkinleştirilmesi

pip komutunu etkinleştirmek için aşağıdaki komutu kullanın.

```
$ python3 -m ensurepip
$ pip3 install --upgrade pip
```

İlk komut python ile gelen pip modülünü çalıştırarak pip kullanmanıza olanak tanır. İkinci komut ise pip sürümünü güncellemek için kullanılır.

Venv ortamı

Python size sistemden bağımsız olarak çalışabileceğiniz sanal bir ortam tanımlama imkanı sağlar. Bu alanı oluşturmak için boş bir dizinde aşağıdaki komutu çalıştıralım.

```
$ python3 -m venv .
```

Ardından bu alanı kullanmaya başlamak için terminale aşağıdaki komutu girelim.

```
$ source bin/activate
```

Bu işlemi yaptıktan sonra terminalimizi kapatana kadarki tüm çalıştıracağımız **pip** komutları ve python ile kurduğumuz paketler **venv** içinden çalışacaktır.

Vala

Vala yazılan kaynak kodu **C** koduna çevirip daha sonra derleyerek çalışan bir programlama dilidir. **valac** kullanılarak derlenir.

Valac kurulumu

Vala derleyicisini (valac) yüklemek için **ymp install vala** komutunu kullanabilirsiniz.

```
$ ymp install vala
```

Kaynak koddan derlemek için ise aşağıdaki adımları takip ediniz:

1. https://gitlab.gnome.org/GNOME/vala adresinden kaynak kodu indirin ve bir dizine çıkarın
2. kaynak kodun içerisine girerek aşağıdaki komutları çalıştırarak derleyin.

```
$ autoreconf -fvi
$ ./configure
$ make
$ make install
```

Nano renklendirme desteği

Kaynak kod yazmak için nano kullanabilirsiniz. Nanoda renklendirme için valaya destek bulunmamaktadır. Vala ile **java** ve **c#** programlama dillerinin renklendirmesi birbirine benzer olduğu için kopyasını kullanarak vala yazarken renklendirme yapabilirsiniz.

Bunun için aşağıdaki yolu izleyin.

1. **/usr/share/nano/** içerisindeki **java.nanorc** dosyasını kopyalayıp aynı dizine **vala.nanorc** olarak kaydedin.
2. Kopyaladığınız dosyayı açın ve java yazan yerleri vala olarak değiştirin.
3. **foreach** ifadesi javada yer almadığı için eksik renklendirilecektir. Bunu nanorc dosyasına elle ekleyebilirsiniz.

C

C en temel programlama dillerinden biridir. C derlemeli bir dil olduğu için **gcc** veya **clang** gibi bir derleyiciye ihtiyaç duyar.

Derleyici kurulumu

Gcc kurulumu

gcc yüklemek için **ymp install gcc** komutunu kullanabilirsiniz.

```
$ ymp install gcc
```

Clang kurulumu

clang yüklemek için **ymp install clang** komutunu kullanabilirsiniz.

```
$ ymp install clang
```

Standart C Kütüphanesi (libc)

Her GNU/Linux dağıtımı bir libc ile gelir. libc, sistemdeki en temel kütüphanedir ve bütün programlar ne ile yazılmış olursa olsun eninde sonunda bir yerlerde libc fonksiyonlarını çağırır. libc fonksiyonlarını kullanmadan program yazmak çok zordur. İşte Türkmen Linux'ta kurulabilen libc implementasyonları:

Glibc

Glibc GNU tarafından geliştirilen ve bakımı yapılan bir libc implementasyonudur ve neredeyse her dağıtım tarafından varsayılan olarak kullanılmaktadır. Türkmen linux için kurmanıza gerek yok çünkü zaten varsayılan olarak gelir ve sistemin çalışması için önemli olduğu için kaldırmaya çalışmanız da pek önerilmez.

Herhangi bir C programını gcc veya clang ile derlediğinizde neredeyse her dağıtımda varsayılan olarak glibc kullanır. Dolayısıyla kullanmak için özel bir çaba sarf etmenize gerek yok.

Musl

Musl, sistemde varsayılan olarak bulunan **glibc** alternatifidir. Musl daha hafiftir fakat dağıtımların geneli glibc kullandığı ve her kaynak kodun düzgün şekilde derlenememesinden dolayı Türkmen linuxta C kütüphanesi olarak glibc tercih edilmiştir.

Yine de musl kullanarak derleme yapılabilmektedir. Bunun için **ymp install musl** komutu ile musl yükleyip sonrasında **musl-gcc** ile derleme yapabilirsiniz.

```
$ ymp install musl
```

Not: tüm sistem kütüphaneleri glibc uyumlu çalıştığı için musl kullanarak yapacağınız derlemelerde sistem kütüphanelerinden yararlanamazsınız.

Derleyici ayarlama

ymp varsayılan olarak **gcc** kullanır. Bunu **/etc/ymp.yaml** içerisinden değiştirebilirsiniz veya **--build:cc=xxx** şeklinde ayarlayabilirsiniz.

Paket yapımı dışında genellikle **CC** çevresel değişkeni kullanılarak derleyici ayarlanabilir.

```
$ export CC=musl-gcc
$ make
```

Rust

Rust llvm tabanlı bir programlama dilidir.

Kurulum

rustup aracını kullanarak kurulum yapılır.

```
$ ymp install rustup
```

Ardından rust yüklenir.

```
# kurulum konumu değiştirmek için
$ export RUSTUP_HOME=$HOME/.local/rust
$ rustup
```

Araçları **PATH** değişkenine ekleyelim

```
export PATH=$PATH:$RUST_HOME/toolchains/stable-$(uname -m)-unknown-linux-gnu/bin
```

Son olarak kontrol etmek için:

```
$ rustc --version
```

Not: rust araçları kullanıcı bazlı olarak kurulur. Sistem bazlı kurulum için öncelikle root yetkisi alarak kurmanız yeterli olur.

Proje oluşturma ve derleme

cargo komutu ile proje oluşturmak için:

```
$ cargo new hello_world
$ cd hello_world
```

Örnek **hello world** kodu aşağıdaki gibi olsun.

```
fn main() {
    println!("Hello, world!");
}
```

Son olarak derleyelim.

```
$ cargo build
```

Çalıştırmak için:

```
$ cargo run
```

Javascript

npm kurulumu

Öncelikle yoksa **~/.profile** dosyasını oluşturun. Ardından **git** komutu yoksa yükleyin.

```
$ touch ~/.profile
$ su
$ ymp install git
```

https://nodejs.org/en/download adresinden **nvm** yüklenir.

~/.profile dosyasını içeri aktaralım ve **npm** yükleyelim

```
$ source ~/.profile
nvm install node
```

Kontrol etmek için *npm --version* komutunu kullanabilirsiniz.

Proje oluşturma

İlk olarak proje dizini oluşturalım.

```
$ mkdir test
```

index.js dosyasını oluşturalım.

```
$ echo 'console.log("Hello World\n");' > index.js
```

Ardından *package.json* dosyamızı aşağıdaki gibi yazalım.

```
{
  "name": "hello-world",
  "version": "1.0.0",
  "description": "Minimal npm project that prints Hello World",
  "main": "index.js",
  "scripts": {
    "start": "node index.js",
  },
  "author": "",
  "license": "MIT"
}
```

Projeyi çalıştırmak için *npm run start* komutunu kullanabiliriz.

Bun kurulumu

npm alternatifi olarak **bun** kullanabilirsiniz. Öncelikle https://bun.sh/ adresinden bun yüklenir.

```
$ curl https://bun.sh/install | bash
```

Ardından gerekli çevresel değişkenler eklenir.

```
$ echo 'export BUN_INSTALL="$HOME/.bun"' >> ~/.profile
$ echo 'export PATH=$BUN_INSTALL/bin:$PATH' >> ~/.profile
```

Proje oluşturma

İlk olarak proje dizini oluşturulur ve *bun init* komutu çalıştırılır.

```
$ mkdir test
$ cd test
$ bun init
```

Ardından **package.json** dosyasına aşağıdaki gibi ekleme yapın:

```
"scripts": {
    "start": "bun run ./index.js",
    "build": "bun build ./index.js --compile --outfile hello-world",
  },
```

Projeyi çalıştırmak için *bun run start* komutunu kullanabilirsiniz.

Projeyi derlemek için *bun run build* komutunu kullanabilirsiniz.

Donanım

Sürücüler

Linux Firmware

linux-firmware linux çekirdeği ile gelmeyen sürücüleri içerir. Bu sürücüler sayesinde wifi gibi bazı ek donanımlar çalışabilmektedir.

linux-firmware kurulumu

linux-firmware Türkmen deposunda paket olarak yer almaz. Bunun yerine **mklinux** paketi içerisinde yer alan **mkfw** komutu yardımı ile kurulur. Bunun için incelikle mklinux kuralım.

```
$ ymp install mklinux
```

Ardından **mkfw** ile linux-firmware kurulumun gerçekleştirelim.

```
$ mkfw -i
```

Not: Alternatif olarak https://git.kernel.org/pub/scm/linux/kernel/git/firmware/linux-firmware.git adresinden arşivi indirin. Bir dizine açın. Ardından **make install** komutu ile kurun.

Son olarak initramfs imajımızı güncellememiz gerekebilir. Bunun için aşağıdaki komutu kullanabiliriz.

```
$ update-initramfs -u -k <kernel-sürümü>
```

DKMS

DKMS (Dynamic Kernel Module Support) Linux çekirdeği için dinamik sürücü desteği sağlayan bir sistemdir. Linux çekirdeğine yerleşik olan modüller dışında kalan, yani çekirdek üzerinde doğrudan derlenmeyen sürücülerin yönetimini sağlar.

Kullanımı

Örneğin /mnt/test dizininde örnek bir kernel modülü olsun.

Bir dkms paketini eklemek için:

```
# modül dizinimizi /usr/src içine kopyalayalım
cp /mnt/test /usr/src/test
# modülü ekleyelim:
dkms add /usr/src/test
```

Eklediğimiz modülleri listelemek için:

```
dkms status
```

Eklediğimiz modülü derlemek için:

```
# derlemek istediğimiz modülün adı ve sürümünü gereklidir.
# Mevcut modüller /var/lib/dkms/ içerisinde bulunur.
```

```
# Bu örnekte test 1.0 olduğunu var sayalım.
dkms build -m test -v 1.0
```

derlediğimiz modülü yüklemek için:

```
# zaten var olan bir modülse --force kullanarak üzerine yazabilirsiniz.
dkms install -m test -v 0.1 --force
```

Kurulan modülü yüklemek için:

```
# öncelikle modül bağımlılık ağacını güncelleyelim.
depmod -a
# modülü yükleyelim
modprobe test
```

Çekirdek güncellendiğinde modülleri otomatik olarak derkelem için:

```
# Bu komutu paket sisteminiz otomatik olarak çalıştırıyor olabilir.
dkms autoinstall
```

Modülü kaldırmak için:

```
# önce modülü kapatalım
modprobe -r test
# modülü silelim
dkms remove -m test -v 0.1
```

dkms paketi hazırlama

Aşağıdaki gibi bir örnek kernel modülümüz olsun:

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Your Name");

MODULE_DESCRIPTION("Example Module");
MODULE_VERSION("0.1");

static int __init test_start(void) {
    printk(KERN_INFO "Hello world\n");
    return 0;
}

static void __exit test_end(void) {
    printk(KERN_INFO "Goodbye\n");
}

module_init(test_start);
module_exit(test_end);
```

Bu modülü derlemek için aşağıdaki **Makefile** dosyası kullanılır:

```
obj-m = test.o
KERNELVER := $(shell uname -r)
all:
	make -C /lib/modules/$(KERNELVER)/build/ M=$(PWD) modules
clean:
	make -C /lib/modules/$(KERNELVER)/build M=$(PWD) clean
```

Şimdi bu kernel modülüne aşağıdaki gibi **dkms.conf** dosyası oluşturabiliriz:

```
PACKAGE_NAME="test"
PACKAGE_VERSION="0.1"
CLEAN="make clean"
MAKE[0]="make all KERNELVER=$kernelver"
BUILT_MODULE_NAME[0]="test"
DEST_MODULE_LOCATION[0]="/updates"
AUTOINSTALL="yes"
```

Örnek projemizin yapısı aşağıdaki gibidir:

```
test-0.1
├── Makefile
├── dkms.conf
└── test.c
```

Ekran Kartı

3D controller kapatmak

Çift ekran kartı bulunan laptoplarda **3D controller** bulunur. Bunun yanında bir de **VGA controller** bulunur.

3D controller daha yüksek güç tükettiği ve gündelik kullanımda hiçbir işe yaramadığı için kapatmak istenilebilir.

Bunun için öncelikle **3D controller** aygıtımızı tespit edelim.

```
for dir in /sys/class/drm/card[0-9*] ; do
# 03 Display controller
# 02 3D controller
    if grep "^0x0302" $dir/device/class ; then
      pci="$(basename $(readlink $dir/device))"
      echo "Found 3D controller: ${pci:5}"
    fi
done
```

Daha sonra bulduğumuz pci adını kullanarak donanımı kapatalım. Bu komutu başlangıçta çalıştırmamız gereklidir.

```
echo 1 > /sys/bus/pci/devices/0000:<pci-adi>/remove
```

Not: Bu işlemi grafik arabirim açıkken yaptığınızda grafik arabirimde beklenmedik sorunlar oluşabilir. Bu sebeple bu işlemi grafik arabirimden servisi başlamadan önce çalıştırılmalıdır.

Not: Türkmen linuxta bu iş için **disable-secondary-gpu** adında bir servis oluşturulmuştur. Bu servisi etkinleştirmeniz yeterli olmaktadır.

```
$ rc-update add disable-secondary-gpu
```

Nvidia

Nvidia ekran kartı sürücüsü olarak genelikle özgür sürücü olan **nouveau** bulunur. Bu sürücü her ne kadar özgür yazılım olsa da sahipli sürücü kadar iyi çalışamayabilir.

Nouveau yüklemek için **mesa** paketini kullanmanız yeterlidir. Çoğu dağıtımda varsayılan olarak nouveau kurulu gelmektedir.

Nouveau sürücüsünü kapatmak için aşağıdaki yolu izleyebiliriz. **/etc/modprobe.d/blacklist-nvidia-nouveau.conf** dosyasına aşağıkini yazalım:

```
blacklist nouveau
options nouveau modeset=0
```

Ardından initramfs dosyamızı güncelleyelim:

```
$ update-initramfs -u -k <kernel-sürümü>
```

Daha sonra sistemi yeniden başlatabiliriz.

Nvidia sürücünün yüklenmesi

Uyarı: Nvidia sürücü özgür değildir. Eğer özgür sürücü kullanmak isterseniz **nouveau** kullanmalısınız.

Not: Bu kısım ile ilgili olarak Archlinux paketine bakabilirsiniz.

Öncelikle Nvidia indirme sayfasından sürücümüzü indirelim. İşletim sistemi olarak **Linux 64-bit** seçmelisiniz.

Daha sonra indirdiğimiz .bin uzantılı dosyayı arşivden çıkartalım. Doküman boyunce indirdiğiniz sürüm numarası yerine **xxxx** koyularak anlatılacaktır. Kendi sürümünüze uygun şekilde değiştirin.

```
sh ~/NVIDIA-Linux-x86_64-xxxx.run --extract-only
```

Şimdi **dkms.conf** dosyamızı yamalayalım.

```
sed -i "s/__VERSION_STRING/xxxx/" kernel/dkms.conf
sed -i 's/__JOBS//' kernel/dkms.conf
sed -i 's/__DKMS_MODULES//' kernel/dkms.conf
sed -i '$iBUILT_MODULE_NAME[0]="nvidia"\
DEST_MODULE_LOCATION[0]="/kernel/drivers/video"\
BUILT_MODULE_NAME[1]="nvidia-uvm"\
DEST_MODULE_LOCATION[1]="/kernel/drivers/video"\
BUILT_MODULE_NAME[2]="nvidia-modeset"\
DEST_MODULE_LOCATION[2]="/kernel/drivers/video"\
BUILT_MODULE_NAME[3]="nvidia-drm"\
DEST_MODULE_LOCATION[3]="/kernel/drivers/video"\
BUILT_MODULE_NAME[4]="nvidia-peermem"\
DEST_MODULE_LOCATION[4]="/kernel/drivers/video"' kernel/dkms.conf
```

İlk olarak nouveau kapatalım ve modprobe ayarımızı ekleyelim.:

```
# Bu kısım nouveau kapatmak için gerekli
echo "blacklist nouveau" > "/etc/modprobe.d/blacklist-nvidia-nouveau.conf"
echo "options nouveau modeset=0" >> "/etc/modprobe.d/blacklist-nvidia-nouveau.conf"
# Bu kısım nvidia için gerekli
echo "nvidia-uvm" >> "/etc/modprobe.d/blacklist-nvidia-nouveau.conf"
```

Şimdi kernel için dkms modülünü kopyalayalım:

```
cp -dr --no-preserve='ownership' kernel /usr/src/nvidia-xxxx"
```

X11 sürücüsünü kopyalayalım:

```
install -Dm755 nvidia_drv.so "/usr/lib/xorg/modules/drivers/nvidia_drv.so"
```

Firmware kopyalayalım:

```
install -Dm644 -t "/lib/firmware/nvidia/xxxx/" firmware/*.bin
```

Kütüphaneleri kopyalayalım:

```
# Wayland/GBM
install -Dm755 libnvidia-egl-gbm.so.1* -t "/usr/lib/"
install -Dm644 15_nvidia_gbm.json "/usr/share/egl/egl_external_platform.d/15_nvidia_gbm.json"
mkdir -p "/usr/lib/gbm"
ln -sr "/usr/lib/libnvidia-allocator.so.xxxx" "/usr/lib/gbm/nvidia-drm_gbm.so"

# OpenGL libraries
install -Dm755 "libEGL_nvidia.so.xxxx" "/usr/lib/libEGL_nvidia.so.xxxx"
install -Dm755 "libGLESv1_CM_nvidia.so.xxxx" "/usr/lib/libGLESv1_CM_nvidia.so.xxxx"
install -Dm755 "libGLESv2_nvidia.so.xxxx" "/usr/lib/libGLESv2_nvidia.so.xxxx"
install -Dm644 "10_nvidia.json" "/usr/share/glvnd/egl_vendor.d/10_nvidia.json"

# OpenGL core library
install -Dm755 "libnvidia-glcore.so.xxxx" "/usr/lib/libnvidia-glcore.so.xxxx"
install -Dm755 "libnvidia-eglcore.so.xxxx" "/usr/lib/libnvidia-eglcore.so.xxxx"
install -Dm755 "libnvidia-glsi.so.xxxx" "/usr/lib/libnvidia-glsi.so.xxxx"

# misc
install -Dm755 "libnvidia-api.so.1" "/usr/lib/libnvidia-api.so.1"
install -Dm755 "libnvidia-fbc.so.xxxx" "/usr/lib/libnvidia-fbc.so.xxxx"
install -Dm755 "libnvidia-encode.so.xxxx" "/usr/lib/libnvidia-encode.so.xxxx"
install -Dm755 "libnvidia-cfg.so.xxxx" "/usr/lib/libnvidia-cfg.so.xxxx"
install -Dm755 "libnvidia-ml.so.xxxx" "/usr/lib/libnvidia-ml.so.xxxx"
install -Dm755 "libnvidia-glvkspirv.so.xxxx" "/usr/lib/libnvidia-glvkspirv.so.xxxx"
install -Dm755 "libnvidia-allocator.so.xxxx" "/usr/lib/libnvidia-allocator.so.xxxx"
install -Dm755 "libnvidia-gpucomp.so.xxxx" "/usr/lib/libnvidia-gpucomp.so.xxxx"

# Vulkan ICD
install -Dm644 "nvidia_icd.json" "/usr/share/vulkan/icd.d/nvidia_icd.json"
install -Dm644 "nvidia_layers.json" "/usr/share/vulkan/implicit_layer.d/nvidia_layers.json"

# VDPAU
```

```
install -Dm755 "libvdpau_nvidia.so.xxxx" "/usr/lib/vdpau/libvdpau_nvidia.so.xxxx"

# nvidia-tls library
install -Dm755 "libnvidia-tls.so.xxxx" "/usr/lib/libnvidia-tls.so.xxxx"
```

```
# CUDA
install -Dm755 "libcuda.so.xxxx" "/usr/lib/libcuda.so.xxxx"
install -Dm755 "libnvcuvid.so.xxxx" "/usr/lib/libnvcuvid.so.xxxx"
install -Dm755 "libcudadebugger.so.xxxx" "/usr/lib/libcudadebugger.so.xxxx"

# NVVM Compiler library loaded by the CUDA driver to do JIT link-time-optimization
install -Dm644 "libnvidia-nvvm.so.xxxx" "/usr/lib/libnvidia-nvvm.so.xxxx"

# PTX JIT Compiler (Parallel Thread Execution (PTX) is a pseudo-assembly language for CUDA)
install -Dm755 "libnvidia-ptxjitcompiler.so.xxxx" "/usr/lib/libnvidia-ptxjitcompiler.so.xxxx"

# raytracing
install -Dm755 "nvoptix.bin" "/usr/share/nvidia/nvoptix.bin"
install -Dm755 "libnvoptix.so.xxxx" "/usr/lib/libnvoptix.so.xxxx"
install -Dm755 "libnvidia-rtcore.so.xxxx" "/usr/lib/libnvidia-rtcore.so.xxxx"

# NGX
install -Dm755 nvidia-ngx-updater "/usr/bin/nvidia-ngx-updater"
install -Dm755 "libnvidia-ngx.so.xxxx" "/usr/lib/libnvidia-ngx.so.xxxx"
install -Dm755 _nvngx.dll "/usr/lib/nvidia/wine/_nvngx.dll"
install -Dm755 nvngx.dll "/usr/lib/nvidia/wine/nvngx.dll"

# Optical flow
install -Dm755 "libnvidia-opticalflow.so.xxxx" "/usr/lib/libnvidia-opticalflow.so.xxxx"

# Cryptography library wrapper
install -Dm755 "libnvidia-pkcs11.so.xxxx" "/usr/lib/libnvidia-pkcs11.so.xxxx"
install -Dm755 "libnvidia-pkcs11-openssl3.so.xxxx" "/usr/lib/libnvidia-pkcs11-openssl3.so.xxxx"

# Debug
install -Dm755 nvidia-debugdump "/usr/bin/nvidia-debugdump"

# nvidia-xconfig
install -Dm755 nvidia-xconfig "/usr/bin/nvidia-xconfig"
install -Dm644 nvidia-xconfig.1.gz "/usr/share/man/man1/nvidia-xconfig.1.gz"

# nvidia-bug-report
install -Dm755 nvidia-bug-report.sh "/usr/bin/nvidia-bug-report.sh"

# nvidia-smi
install -Dm755 nvidia-smi "/usr/bin/nvidia-smi"
install -Dm644 nvidia-smi.1.gz "/usr/share/man/man1/nvidia-smi.1.gz"

# nvidia-cuda-mps
install -Dm755 nvidia-cuda-mps-server "/usr/bin/nvidia-cuda-mps-server"
install -Dm755 nvidia-cuda-mps-control "/usr/bin/nvidia-cuda-mps-control"
install -Dm644 nvidia-cuda-mps-control.1.gz "/usr/share/man/man1/nvidia-cuda-mps-control.1.gz"

# nvidia-modprobe
install -Dm4755 nvidia-modprobe "/usr/bin/nvidia-modprobe"
install -Dm644 nvidia-modprobe.1.gz "/usr/share/man/man1/nvidia-modprobe.1.gz"

# nvidia-persistenced
install -Dm755 nvidia-persistenced "/usr/bin/nvidia-persistenced"
install -Dm644 nvidia-persistenced.1.gz "/usr/share/man/man1/nvidia-persistenced.1.gz"
```

Son olarak dkms modülümüzü derleyelim ve initramfs güncelleyelim:

```
$ dkms autoinstall
$ update-initramfs -u -k <kernel-sürümü>
```

Sistemi yeniden başlatabilirsiniz.

Çoklu ekran kartı kullanımı

Eğer cihazınızda nvidia 2. ekran kartı ise varsayılan olarak uygulamalar nvidia ile çalışmaz. Bunun için uygulamayı aşağıdaki gibi çalıştırmalısınız.

```
__NV_PRIME_RENDER_OFFLOAD=1 __GLX_VENDOR_LIBRARY_NAME=nvidia <çalıştırılacak-komut>
```

Eğer tüm sistemin nvidia ile çalışmasını istiyorsanız Bu değişkenleri **/etc/profile.d/nvidia.sh** içerisine kaydedin.

```
export __NV_PRIME_RENDER_OFFLOAD=1
export __GLX_VENDOR_LIBRARY_NAME=nvidia
```

Not: Bu yöntem sağlıklı çalışmayabilir.

Çoklu ekran kartı

Eğer birden çok ekran kartınız varsa varsayılan olarak ilk ekran kartı kullanılır. Diğer ekran kartınızı kullanmak için **DRI_PRIME** çevresel değişkeni ayarlamamız gerekir.

Öncelikle ekran kartlarını listeleyelim:

```
$ lspci -nn | grep -ie vga -ie 3d
# X11 kullanıyorsanız
$ xrandr --listproviders
```

İkinci ekran kartını kullanmak için uygulamayı aşağıdaki gibi çalıştıralım.

```
# 1 numaralı ekran kartı (2. ekran kartı) için:
$ DRI_PRIME=1 firefox
# Veya vendor:product şeklinde de belirtebilirsiniz
$ DRI_PRIME=1002:699f firefox
```

Not: Vulkan ile çalışan uygulamalarda da kullanıma zorlamak için **DRI_PRIME=1!** şeklinde tanımlanır.

Eğer sürekli olarak belirlediğiniz ekran kartını kullanmak isterseniz bu değişkeni **/etc/profile.d/** dizini altında tanımlamanız yeterli olur.

```
echo "export DRI_PRIME=1" > /etc/profile.d/dri_prime.sh
```

Not: Bu işlem grafik arabirimde çeşitli sorunlara ve aşırı güç kullanımına sebep olabilir.

Çoklu Ortam

Pipewire

Pipewire çoklu ortam yöneticisidir.

Türkmen linuxta pipewire pulseaudio yerine kullanılır ve pipewire-pulse pakete dahildir.

Kurulumu

pipewire paketini yüklemeniz gerekmektedir. Ardından **wireplumber** yükleyip masaüstü ortamı ile çalışmasını sağlamalısınız.

```
$ ymp install pipewire wireplumber
```

Türkmen linuxta pipewire sistem servisi olarak çalışmaktadır. Bu sebeple sistem servislerini çaşılışa eklemelisiniz.

```
# başlangıçta çalıştırmak için
$ rc-update add pipewire default
$ rc-update add pipewire-pulse default
$ rc-update add wireplumber default
# başlatmak için
$ rc-service pipewire start
$ rc-service pipewire-pulse start
$ rc-service pipewire start
```

Kurulum tamamlandıktan sonra oturumunuzu kapatıp açmanız gerekebilir.

Çalışıp çalışmadığını test etmek için **pactl info** ve **wpctl status** komutlarını kullanarak pipewire ile ilgili bilgi alabilirsiniz.

Not: pactl komutu pipewire-pulse ile sağlanır. wpctl komutu wireplumber ile sağlanır.

Uzak makinaya bağlanma

Öncelikle gerekli modülü sunucu olarak kullanılacak makinada aşağıdaki komut ile etkinleştirelim.

```
# Bu kısım sesi alacak olan makinada çalıştırılır.
# auth-ip-acl parametresini yazmazsanız herkes tarafından elişilebilir olur.
# Sadece bu parametre ile belirtilen ip adresine izin verilir.
# Birden çok ip belirtmek için aralarına ; işareti koyulmalıdır.
$ pactl load-module module-native-protocol-tcp auth-ip-acl=192.168.0.18;192.168.0.15
```

Daha sonra bağlantı kurmak için **PULSE_SERVER** çevresel değişkenini kullanabiliriz.

```
# Bu kısım sesi gönderecek olan makinada çalıştırılır.
$ export PULSE_SERVER=192.168.0.12
$ mpv /home/pingu/test.mkv
```

Ses seviyesi ayarı

Wireplumber üzerinden ses seviyelerini ayarlayabilirsiniz.

Öncelikle **wpctl status** komutu ile mevcut ses aygıtlarının ve uygulamaların id değerlerini bulalım.

```
PipeWire 'pipewire-0' [0.3.67, root@(none), cookie:2586580591]
 └─ Clients:
        31. pipewire-pulse                      [0.3.67, root@(none), pid:4772]
        33. WirePlumber                         [0.3.67, root@(none), pid:4771]
        34. WirePlumber [export]                [0.3.67, root@(none), pid:4771]
        63. Firefox                             [0.3.67, root@(none), pid:4806]
        69. wpctl                               [0.3.67, root@(none), pid:9322]

Audio
 ├─ Devices:
 │      47. Built-in Audio                      [alsa]
```

```
  │      55. Built-in Audio                      [alsa]
  │      59. Built-in Audio                      [alsa]
  │
  ├─ Sinks:
  │      32. Built-in Audio Analog Stereo      [vol: 0.39]
  │      51. Built-in Audio Analog Stereo      [vol: 0.40]
  │    *   57. Built-in Audio Analog Stereo      [vol: 1.00]
  │
  ├─ Sink endpoints:
  │
  ├─ Sources:
  │      37. Built-in Audio Analog Stereo      [vol: 1.00]
  │      38. Built-in Audio Stereo             [vol: 1.00]
  │    *   52. Built-in Audio Analog Stereo      [vol: 1.00]
  │
  ├─ Source endpoints:
  │
  └─ Streams:
         64. Firefox
              44. output_FL       > Generic Analog:playback_FL   [active]
              66. output_FR       > Generic Analog:playback_FR   [active]

Video
 ├─ Devices:
 │
 ├─ Sinks:
 │
 ├─ Sink endpoints:
 │
 ├─ Sources:
 │
 ├─ Source endpoints:
 │
 └─ Streams:

Settings
 └─ Default Configured Node Names:
```

Bulduğumuz id değerini kullanarak ses seviyesini ayarlayabiliriz.

```
# Ses seviyesi 0-1 arası değerde olmalıdır.
# Daha yüksek seviyeler de ayarlanabilir. (tavsiye edilmez)
$ wpctl set-volume 57 0.8
```

Sessiz moda alıp geri açmak için aşağıdaki gibi komut kullanılabilir.

```
# 1 sessiz moda alır. 0 sessiz moddan çıkar.
$ wpctl set-mute 57 0
```

Güç Kontrolü

Uyku modu

Linux dağıtımlarında uyku modu **elogind** tarafından gerçekleştirilir. Uyku moduna almak için öncelikle desteklenen modları kontrol etmelisiniz.

```
$ cat /sys/power/state
>> freeze mem disk standby
```

Buradaki modlur şu anlama gelir:

- **freeze**: bekleme modu
- **mem**: uyku modu
- **disk**: hazırda beklet modu
- **stanby**: güç açık bekleme modu

Not: bazı donanımlarda bu modların bir kısmı bulunmayabilir.

Ardından bilgisayarınız linux için acpi tablosuna sahip olduğunu kontrol edin.

```
$ strings /sys/firmware/acpi/tables/DSDT | grep linux -i
>> Linux
```

Yukarıdaki komut boş çıktı veriyorsa acpi tablosu yok anlamına gelir. Linux diğer acpi tabloları ile çalışmayı dener ve muhtemel sorunlara sebep olabilir.

Uyku modunu tetikleme

logind kullanarak aşağıdaki gibi uyku moduna alabilirsiniz.

```
$ loginctl suspend
```

Veya doğrudan kernel üzerinden uyku moduna geçirebilirsiniz.

```
$ echo mem > /sys/power/state
```

Not: Bazı donanımlar uyku modunda düzgün çalışmayabilir.

Uyku moduna geçmeyi önleme

Uyku moduna geçmeyi önlemek için **/etc/elogind/suspend.conf** dosyasını aşağıdaki gibi düzenlemelisiniz.

```
[Sleep]
AllowSuspend=yes
```

Laptop kapağı kapatma eylemi

Eğer bir laptop kullanıyorsanız ve kapağı kapattığınızda uykuya geçmesini istemiyorsanız onun için de **/etc/elogind/logind.conf** dosyasını aşağıdaki gibi düzenlemelisiniz.

```
[Login]
HandleLidSwitch=ignore
```

```
HandleLidSwitchExternalPower=ignore
HandleLidSwitchDocked=ignore
```

Burada **ignore** yerine: **poweroff**, **reboot**, **suspend** yazarak ilgili eylemi yapmasını sağlayabilirsiniz.

Güç tasarrufu

Linux dağıtımları varsayılan ayarlarda sunucu gibi çalışmak için ayarlanmıştır. Güç kullanımı açısından herhangi bir kısıtlama olmadığı için güç tasarrufu yapılmaz. Fakat güç tasarrufunu kendiniz ayarlamanız mümkündür.

İşlemci Governor Ayarları

İşlemci çeşitli güç modlarına (governor) sahiptir. Bu modlar sayesinde duruma göre hızlı çalışma veya daha az güç harcama arasında seçim yapılabilir.

Kullanılabilir modları görmek için:

```
$ cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_available_governors
> conservative ondemand userspace powersave performance schedutil
```

Mevcut seçili olan modu görmek için:

```
$ cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
> schedutil
```

Modu değiştirmek için:

```
$ echo "powersave" > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
```

Bu şekilde **governor** değiştirerek güç tasarrufu sağlanabilir.

İşlemci çekirdeklerini uykuya almak

Sistemde bulunan işlemci çekirdeklerini uykuya almak daha az güç kullanımını sağlayabilir.

Bunun için öncelikle kaç çekirdek olduğunu bulalım.

```
$ nproc
> 24
```

Şimdi aşağıdaki gibi çekirdeği kapatabiliriz. Örneğin 3. çekirdeği kapatalım:

```
# kapatmak için 0 açmak için 1
$ echo 0 > /sys/devices/system/cpu/cpu3/online
```

Not: 0. çekirdek kapatılamaz.

Not: 4 veya daha az çekirdeğin açık kalması sisteminizin performansını kötü etkileyebilir.

Platform profile ayarı

Bazı donanımlar güç tasarrufu donanımsal profili bulundurur. Eğer mevcutsa aşağıdaki gibi kullanabilirsiniz.

```
# düşük güç
echo "low-power" > /sys/firmware/acpi/platform_profile
# normal güç
echo "balanced" > /sys/firmware/acpi/platform_profile
# yüksek güç
echo "performance" > /sys/firmware/acpi/platform_profile
```

Turbo boost ayarı

İşlemciniz turbo boost destekliyorsa linuxta varsayılan olarak açık olarak gelmektedir. Bu yüksek güç kullanımına sebep olabilir Ayağıdaki gibi kapatabilirsiniz:

```
# kapatmak için
echo 1 > /sys/devices/system/cpu/intel_pstate/no_turbo
echo 0 > /sys/devices/system/cpu/cpufreq/boost
# açmak için
echo 0 > /sys/devices/system/cpu/intel_pstate/no_turbo
echo 1 > /sys/devices/system/cpu/cpufreq/boost
```

ACPI

Acpi (Advanced Configuration and Power Interface) bilgiayarınızın güç yönetimini ve donanımını kontrol etmenizi sağlayan bir arayüz standartıdır. Acpi uykuya alma güç tasarrufu ve benzeri işlemleri kontrol eder.

Acpi tabloları

Bilgisayarınızın acpi tabloları bulunur ve işletim sistemi buraya bakarak çalışır. Bilgisayarınız doğrudan linux için acpi tablosuna sahip olmayabilir. Bu durumda linux diğer sistemlere ait acpi tablolarını kullanmaya çalışır. Bu durum bilgisayarınızın kararlı çalışmamasına sebep olabilir.

Bilgisayarınızın linux için bir ACPI tablosuna sahip olup olmadığını anlamak için aşağıdaki komutu kullanabilirsiniz.

```
if grep -i linux /sys/firmware/acpi/tables/DSDT 2>/dev/null ; then
    echo "Acpi linux destekliyor"
else
    echo "Acpi linux desteklemiyor"
fi
```

Ayrıca bilgisayarınızın bir oem anahtarı varsa acpi tablosundan öğrenebilirsiniz. Eğer bir oem bulunmuyorsa aşağıdaki dizin bulunmamaktadır.

```
tail -c 32 /sys/firmware/acpi/tables/MSDM && echo
```

Not: OEM bulunan bilgisayarlar acpi sorunlarına sahip olabilir. Bilgisayar satın alırken oem bulunmamasına dikkat etmelisiniz.

Acpi wakeup ayarları

Bilgisayarınızı uyku moduna aldığınızda veya boşta bıraktığınızda enerji tasarrufu amaçlı bazı donanımlar kapatılabilir. Bu durum iyi olsa da bazı durumlarda uykudan uyanamama gibi sorunlara sebep olabilir.

Hangi kesmelerin açık olduğunu görmek için **/proc/acpi/wakeup** içerisine bakabiliriz.

```
cat /proc/acpi/wakeup
> Device        S-state   Status   Sysfs node
> GP12    S4    *enabled   pci:0000:00:07.1
> GP13    S4    *enabled   pci:0000:00:08.1
> XHC0    S4    *enabled   pci:0000:0b:00.3
> GP30    S4    *disabled
> GP31    S4    *disabled
> PS2K    S3    *disabled
> PS2M    S3    *disabled
> X161    S4    *disabled
> SWUS    S4    *enabled   pci:0000:07:00.0
> SWDS    S4    *enabled   pci:0000:08:00.0
> X162    S4    *disabled
> PTXH    S4    *enabled   pci:0000:01:00.0
> X1_1    S4    *disabled
> WIFI    S4    *disabled  pci:0000:05:00.0
> RLAN    S4    *enabled   pci:0000:06:00.0
> X162    S4    *disabled
> M2_2    S4    *disabled  pci:0000:04:00.0
```

Bir argıtın kesmesinin durumunu değiştirmek için aygıtın adını bu dosyaya yazabiliriz.

Ekran parlaklığı kontrolü

Ekran parlaklığı kontolü ile ilgili sorun yaşıyorsanız kontrol yöntemini değiştirebilirsiniz.

Bunun için çekirdeğe **acpi_backlight** parametresi eklemeniz gereklidir. Bu parametre şu değerleri alabilir.

vendor

Bu değer, ekran parlaklığını kontrol etmek için üreticiye özgü yöntemleri kullanır. Genellikle, bu yöntem dizüstü bilgisayarların üreticileri tarafından sağlanan özel sürücülerle birlikte çalışır.

video

Bu değer, video sürücüsünü kullanarak ekran parlaklığını kontrol eder. Genellikle, bu yöntem genel bir çözüm olarak kabul edilir ve birçok sistemde çalışır.

native

Bu değer, ACPI'nin yerel yöntemlerini kullanarak ekran parlaklığını kontrol eder.

Ekran parlaklığı

Bir laptop kullanıyorsanız ve ekran parlaklığını elle kontrol etmek istiyorsanız bu bölümde nasıl yapılacağı anlatılmıştır.

öncelikle hangi aygıtlar mevcut listeleyelim

```
ls /sys/class/backlight
> intel_backlight
```

Ardından maximum değeri öğrenelim.

```
cat /sys/class/backlight/intel_backlight/max_brightness
> 24242
```

Şimdi istediğimz değeri ayarlayabiliriz. Bunun için istediğimz değeri ağaıdaki gibi yazalım.

```
# %50 parlaklık için
echo 12000 > /sys/class/backlight/intel_backlight/brightness
```

Not bu işlem için root gereklidir. %0 parlaklık ekranın tamamen kararmasına neden olabilir.

Grafik Arabirim

Uygulamalar

AppImage

AppImage paketleri tüm bağımlılıkları içerisinde kurulu gelen tek dosyadan oluşan uygulama paketleridir. Bu paketler sayesinde uygulamaları sürümden bağımsız şekilde ve ek bağımlılık kurmadan kullanabilirsiniz.

AppImage için fuse yüklemeniz gerekmektedir.

AppImage dosyalarına https://appimage.github.io/ ve https://www.appimagehub.com/ adreslerinden ulaşabilirsiniz.

AppImage çalıştırmak

Öncelikle dosyayı çalıştırılabilir yapmalısınız.

```
$ chmod 755 dosya.appimage
```

Daha sonra dosyayı komut çalıştırır gibi çalıştırabilirsiniz.

```
$ ./dosya.appimage
```

Not: AppImage dosyaları herhangi bir güvenlik kontrolünden geçirilmediği için güvenilir olmayan kaynaklardan gelen dosyaları çalıştırmayınız.

AppImage dosyalarını uygulama menüsüne eklemek

Bunun appimage dosyamısa **--appimage-extract** parametresi vererek içini açalım. Bize **squashfs-root** dizini oluşturulacaktır. Bu dizini istediğimiz bir yere kopyalayıp aşağıdaki gibi bir uygulama başlatıcısı hazırlayalım. Bu başlatıcıyı ~/.local/share/applications dizinine koyalım.

```
[Desktop Entry]
Version=1.0
Name=Firefox
Comment=Web Browser
Exec=/data/user/pingu/firefox/firefox %u
Icon=/data/user/pingu/firefox/icon.png
Terminal=false
Type=Application
MimeType=text/html;
Categories=Network;WebBrowser;
```

Burada MimeType kısmına belirtilen dosyalar birlikte aç menüsünde gözükmesini sağlar. Bir dosyanın mimetype adına **file --mime dosya** komutu ile ulaşabilirsiniz.

Flatpak

Flatpak sistemden bağımsız şekilde uygulama çalıştıran bir altyapıdır. Bu sayede normal yolla kullanmak için çok fazla bağımlılığa ihtiyaç duyan veya kurulması mümkün olmayan uygulamalar çalıştırılabilir.

Kurulum

Öncelikle **flatpak** yükleyelim.

```
$ ymp install build-base --no-emerge
$ ymp install flatpak
```

Daha sonra gereken servisleri etkinleştirelim.

```
$ rc-update add devfs
$ rc-update add fuse
$ rc-update add hostname
```

Depo ekleme

Flatpak için uygulama deposu eklememiz gereklidir. Flathub deposunu aşağıdaki gibi ekleyebilirsiniz.

```
$ flatpak remote-add --user --if-not-exists flathub https://flathub.org/repo/flathub.flatpakrepo
# --user parametresi yetkili kullanıcı olmadan eklemek içindir.
```

İsterseniz aşağıdaki gibi alias tanımı yapıp sürekli kullanıcı modunda kullanabilirsiniz.

```
alias flatpak='flatpak --user'
```

Uygulama yükleme

flatpak search kullanarak uygulama arayabilirsiniz. Yüklemek için **flatpak install** kaldırmak için ise **flatpak remove veya flatpak uninstall** komutları kullanılır.

Not: Yükleme ve kaldırma için **appid** değeri kullanılır.

```
$ flatpak search  dolphin
Name      Description              Application ID      Version   Branch Remotes
Dolphin   File Manager             org.kde.dolphin     23.04.0   stable flathub
...
$ flatpak install org.kde.dolphin
...
$ flatpak remove org.kde.dolphin
...
```

Uygulamaları güncelleme

flatpak upgrade komutu ile güncelleyebilirsiniz.

```
$ flatpak upgrade
```

Bubblewrap

Bubblewrap root yetkisine ihtiyaç duymadan uygulamaları kısıtlı alanda çalıştırabilmemize yarayan bir araçtır.

Türkmen linuxa yüklemek için aşağıdaki komutu kullanabilirsiniz:

```
$ ymp install bubblewrap
```

Kullanımı

Bubblewrap **bwrap** komutu ile kullanılır. Bu komut aldığı parametrelere göre kısıtlı alanınızı oluşturur. Oluşturulan kısıtlı alanda girmiş olduğunuz komut çalıştırılır. Örneğin:

```
$ bwrap --ro-bind / / \
    --bind $HOME $HOME \
    --tmpfs $HOME/gecici
    --dev /dev \
    --proc /proc \
    --bind /sys /sys \
    --setenv HOME $HOME/gecici \
    --hostname sandbox \
    -- /bin/bash -i
```

Yukarıdaki komut ile:

- kök dizin yazılamaz bağlandı
- ev dizin yazılabilir bağlandı
- /dev /sys /proc dizinleri bağlanı
- ev dizini değişkeni ayarlandı
- hostname ayarlandı
- parametre algılama sonlandırıldı.
- /bin/bash çalıştırıldı.

Başka dağıtım ile kullanma

Bwrap kullanarak başka bir linux dağıtımını uygulama çalıştırmak için kullanabiliriz. Bu dokümanda örnek olarak alpine linux üzerinden anlatılacaktır.

İlk olarak rootfs imajını edinelim. Alpine linux için: https://www.alpinelinux.org/downloads/ adresinden minimal rootfs olanı indirin.

Ardından dosyaları bir dizine çıkarın. Bu aşamada root yetkisi kullanmayacağımız için dosya izinlerinin kendi kullanıcınıza ait olması gerekir.

```
$ mkdir alpine
$ cd alpine
$ tar -xf alpine-minirootfs-3.22.2-x86_64.tar.gz
$ cd ..
```

Son olarak içerisine girmek için aşağıdaki gibi bir komut kullanalım. Bu komutu daha sonra kullanmak için bir dosyaya kaydedebilirsiniz.

```
# dbus session bus dizinini bulalım
$ dbus_dir=$(echo $DBUS_SESSION_BUS_ADDRESS | cut -f2 -d"=" | cut -f1 -d",")
# shell çalıştıralım
$ bwrap --bind ./alpine / \
    --dev /dev \
    --proc /proc \
    --bind /sys /sys \
    --dev-bind /dev/dri /dev/dri \
    --tmpfs /tmp \
    --setenv PS1 "\$ (alpine) " \
```

```
    --ro-bind / /system \
    --bind /tmp/.X11-unix /tmp/.X11-unix \
    --bind $XDG_RUNTIME_DIR $XDG_RUNTIME_DIR \
    --bind $dbus_dir $dbus_dir \
    --bind $HOME $HOME \
    -- /bin/ash -i
```

Ardından paket sistemini kullanarak içerisine istediğiniz uygulamayı yükleyebilirsiniz ve çalıştırabilirsiniz. Örneğin:

```
$ (alpine) apk add chromium font-dejavu
$ (alpine) chromium
```

Not: Bu şekilde kullandığınız dağıtım root yetkisi yerine sizin kullanıcınızda çalışacaktır ve bu durum bazı öngörülemez hatalara sebep olabilir.

Tek kullanımlık ev dizini

bwrap kullanarak tek kullanımlık ev dizini oluşturabilir ve uygulamalarınızı boş ortamda deneyebilirsiniz.

```
$ bwrap --ro-bind / / \
    --dev /dev \
    --proc /proc \
    --bind /sys /sys \
    --bind /tmp /tmp \
    --bind /run /run \
    --tmpfs $HOME \
    -- /usr/bin/firefox
```

Araç Setleri

Gtk

Gtk (GIMP Tool Kit) bir özgür grafik araç setidir. C ile yazılmıştır fakat birçok programlama dili ile kullanılabilir.

Örnek olarak aşağıdaki C kodu gösterilebilir. (Gtk 3 için)

```c
#include <gtk/gtk.h>

int main(int argc, char** argv){
    // gtk init
    gtk_init (&argc, &argv);
    // pencere oluşturma
    GtkWindow *window = gtk_window_new();
    // pencereyi gösterme
    gtk_widget_show_all(window);
    // uygulamayı çalıştırma
    gtk_main();
    // çıkış
    return 0;
}
```

Derlemek için aşağıdaki komut kullanılabilir.

```
$ gcc -c main.c `pkg-config --cflags gtk+-3.0`
$ gcc -o main main.o `pkg-config --libs gtk+-3.0`
```

python ile ile aşağıdaki gibi kullanılabilir. (Gtk 3 için)

```
# gobject kütüphanesini yükleme
import gi
gi.require_version('Gtk', '3.0')
from gi.repository import Gtk
# pencere oluşturma
window = Gtk.Window()
# pencereyi gösterme
window.show_all()
# çalıştırma
Gtk.main()
```

Tema yükleme

Simge temalarını **~/.icons** içine, gtk temalarını ise **~/.themes** içine kopyalamanız gerekmektedir. Ana dizinin içerisinde **index.theme** dosyası bulunmalıdır. Alt dizinler dikkate alınmaz.

Tema ayarlama

Gtk temasını tek bir uygulama için değiştirmek için **GTK_THEME** değişkenini değiştirebilirsiniz.

```
$ GTK_THEME=Arc-Dark thunar
```

Tüm uygulamalar için değiştirmek istiyorsanız **~/.config/gtk-3.0/settings.ini** (gtk4 için gtk-4.0) dosyasını düzenlemelisiniz.

```
[Settings]
gtk-theme-name=Arc-Dark
gtk-icon-theme-name=Tango
```

Not: libadwaita tabanlı gtk4 uygulamalarında tema ayarlama konusunda sıkıntı yaşayabilirsiniz. Temayı zorlamak istiyorsanız **GTK_THEME** değişkenini kullanmayı deneyin.

Not: kullandığınız masaüstü ortamının tema ayarları varsa öncelikle onu kullanmalısınız.

Yazar Notu: Göz sağlığınız için açık tema kullanmayın :D

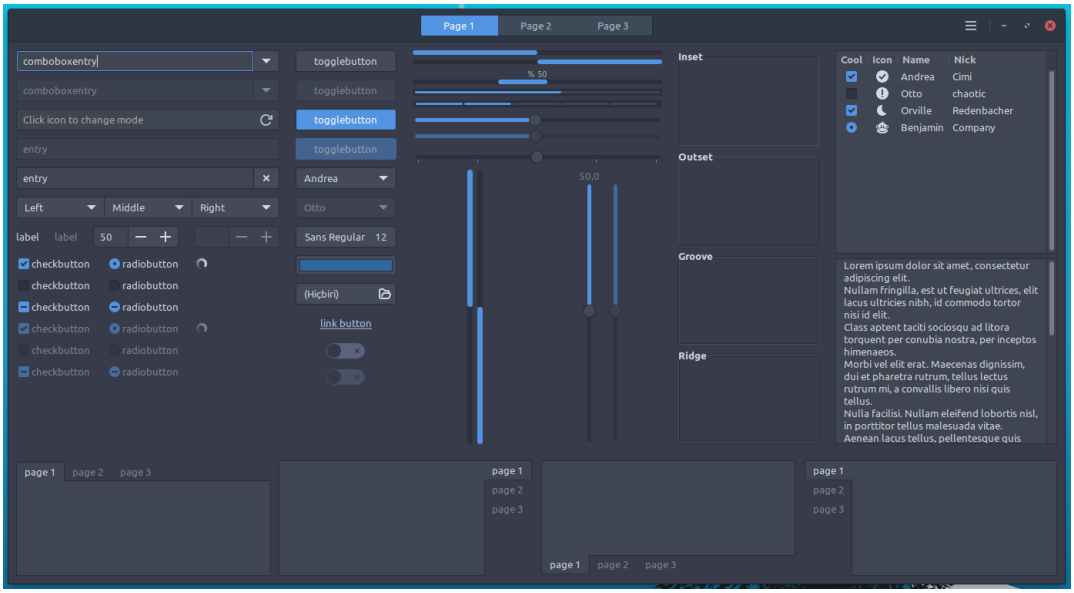
Flatpak uygulamaları temanızı dikkate almıyorsa aşağıdaki komut ile düzeltebilirsiniz:

```
# kullanıcı temaları
$ flatpak override org.example.application --filesystem=~/.themes:ro
# sistem temaları
$ flatpak override org.example.application --filesystem=/usr/share/themes:ro
```

Tema test etme

Temayı test etmek için **widget-factory** kullanabilirsiniz. Gtk3 için:

```
$ gtk3-widget-factory
```



Gdk backend ayarı

GDK_BACKEND değişkenini ayarlayarak uygulamanın X11 veya Wayland üzerinde çalışmasını zorlayabilirsiniz.

Örneğin wayland uyumlu olmayan bir uygulamayı Xwayland ile açmaya zorlamak için:

```
$ GDK_BACKEND=x11 xfce4-panel
```

Overlay Scrollbar kapatma

Gtk3 için **GTK_OVERLAY_SCROLLING=0** değişkenini kullanabilirsiniz.

```
$ GTK_OVERLAY_SCROLLING=0 thunar
```

Tüm uygulamalarda kullanmak için ise **~/.config/gtk-3.0/settings.ini** dosyasına aşağıdaki gibi ekleme yapmalısınız:

```
[Settings]
gtk-overlay-scrolling = false
```

Gtk4 için ise Bu seçenek kullanılabilir değildir. Uygulamayı yamalayıp tekrar derlemeniz gerekebilir.

Dokunma test modunu açma

Eğer dokunmatik ekranlı bir cihaz kullanıyorsanız veya yazacağınız uygulamanın dokunmatik ekranda nasıl çalışacağını test etmek istiyorsanız aşağıdaki değişkenleri ayarlamalısınız.

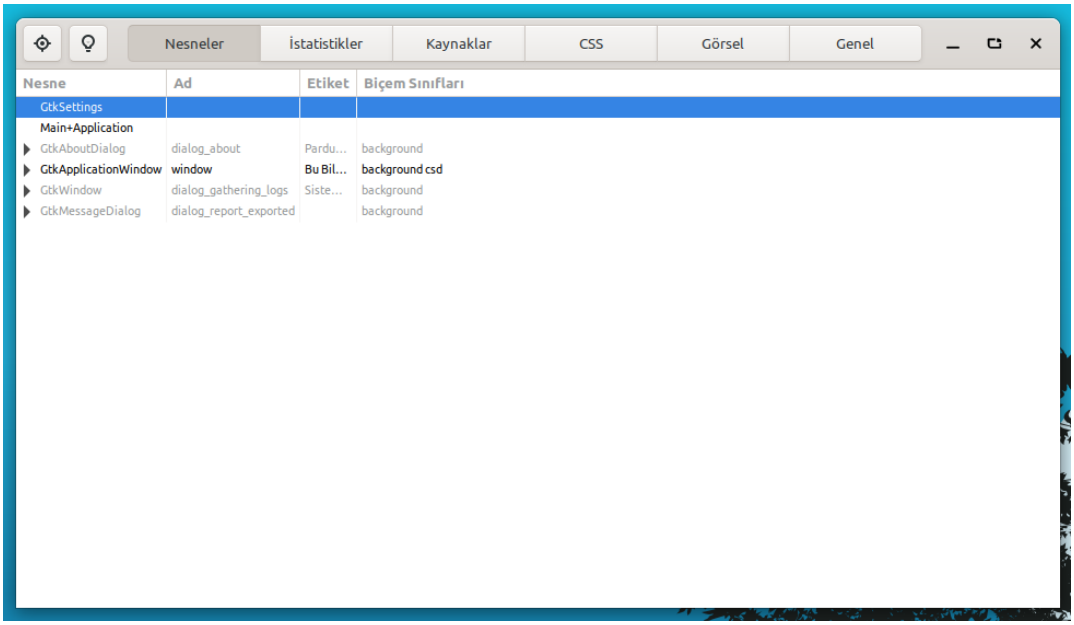
```
# Dokunma test modunu açmak için
export GTK_TEST_TOUCHSCREEN=1
xfce4-appfinder
```

Not: Gtk dokunarak kullanımda çeşitli sorunlar çıkarabilir. Bunun için dokunma eylemlerini kapatıp dokunma test modunu açabilirsiniz. Ana eylemler dışındaki eylemleri kapatmak için:

```
# Sadece ana eylemleri kullanmak için.
export GTK_CORE_DEVICE_EVENTS=1
```

Hata ayıklama

Gtk uygulamalarında hata ayıklama yapmak için **GTK_DEBUG=all** ve **G_MESSAGES_DEBUG=all** çevresel değişkenleri ayarlanır.



Qt

Qt bir özgür grafik araç setidir. C++ ile yazılmıştır ve uyumlu programlama dilleri ile kullanılabilir.

Örneğin aşağıda bir basit bir Qt C++ kodu örneği verilmiştir.

```
#include <QApplication>
#include <QPushButton>

int main(int argc, char *argv[]) {
    // Qapplication oluştur
    QApplication a(argc, argv);
    // Bir button oluştur
    QPushButton button ("Hello world!");
    // Bu button nesnesini göster
    button.show();
    // Uygulamayı başlat
    return a.exec();
}
```

Uygulama derleme

Qt uygulamalarını derlemek için ise **qmake** adı verilen bir derleme talimatından faydalanılır.

```
QT       += core gui

greaterThan(QT_MAJOR_VERSION, 6): QT += widgets

TARGET = helloworld
TEMPLATE = app

SOURCES += main.cpp
```

Bu dosyayı **hello.pro** şeklinde kaydedip şu şekilde derleyebiliriz.

```
$ qmake
$ make
```

Ayrıca qmake yerine şu şekilde elle derlememiz de mümkündür.

```
$ g++ main.cpp -c `pkg-config --cflags Qt6Core Qt6Gui Qt6Widgets`
$ g++ -o main main.o `pkg-config --libs Qt6Core Qt6Gui Qt6Widget`
```

QT_QPA_PLATFORM

Qt uygulamaları varsayılan olarak X11 veya Wayland üzerinde hangisi ile çalışıyorsa ona uygun şekilde pencere oluşturur. Eğer Wayland kullanıyorsanız ve uygulamalın Xwayland ile çalışmasını istiyorsanız aşağıdaki gibi değişken ayarlayabilirsiniz.

```
# önce xcb eğer kullanılabilir değilse wayland ayarlamak için
$ export QT_QPA_PLATFORM=xcb;wayland
$ ./main
# alternatif olarak şu da geçerli olur.
$ ./main -platform xcb;wayland
```

Ayrıca Qt uygulamalarını X11 veya Wayland yerine doğrudan framebuffer üzerinde çalıştırmak da mümkündür. Bunun için **linuxfb** olarak başlatmalısınız.

```
$ QT_QPA_PLATFORM=linuxfb ./main
```

Temalama

Qt uygulamasının gtk ile aynı temayı kullanması için aşağıdaki çevresel değişkeni kullanabilirsiniz:

```
$ export QT_QPA_PLATFORMTHEME=gtk3
```

Masaüstü Ortamları

LXDE

Lxde hafif masaüstü ortamıdır. Az kaynak kullanması ve nadiren güncelleme aldığı için tercih edilir.



Kurulumu

Ymp kullanarak lxde yüklemek için aşağıdaki komutu kullanabilirsiniz.

```
$ ymp install @lxde
```

Lxde openbox pencere yöneticisi ile çalışır. Bu yüzden openbox yüklemeniz gerekmektedir.

```
$ ymp install openbox
```

xinitrc ayarları

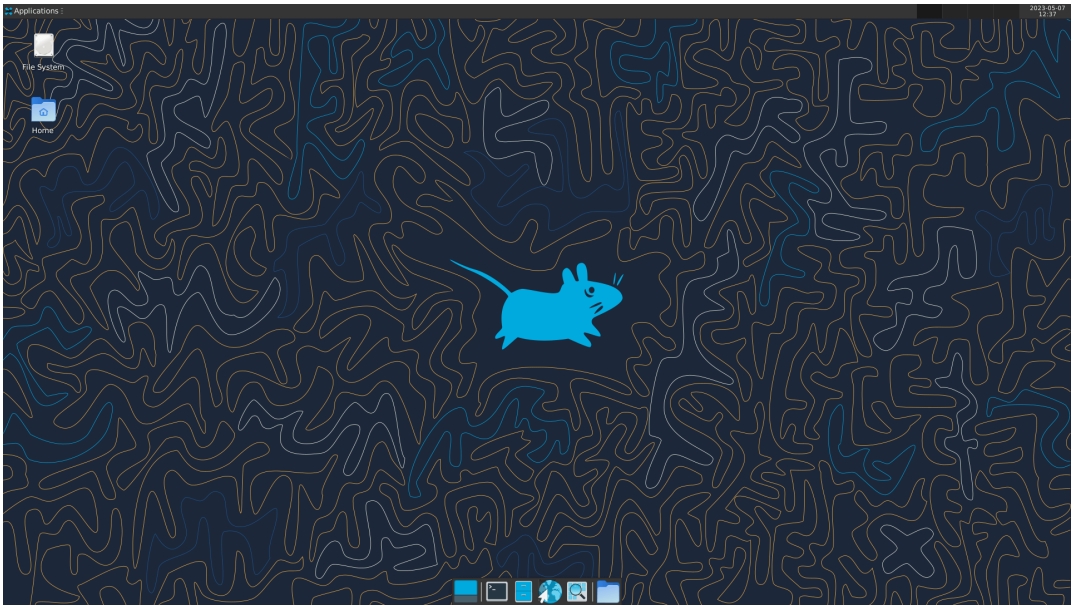
startx komutu çalıştırdığınızda lxde başlatılmasını istiyorsanız **~/.xinitrc** dosyanızın sonuna aşağıdaki gibi ekleme yapmalısınız.

```
exec startlxde
```

Bununla birlikte elogind gerektiği için elogind etkinleştirmelisiniz.

Xfce

Xfce kolay kullanılan ve en çok tercih edilen masaüstü ortamlarından biridir.



Kurulumu

Ymp kullanarak xfce yüklemek için aşağıdaki komutu kullanabilirsiniz.

```
$ ymp install @xfce.base
```

Ek paketleri yüklemek için ise **@xfce.extra** yüklemelisiniz.

xinitrc ayarları

startx komutu çalıştırdığınızda xfce başlatılmasını istiyorsanız **~/.xinitrc** dosyanızın sonuna aşağıdaki gibi ekleme yapmalısınız.

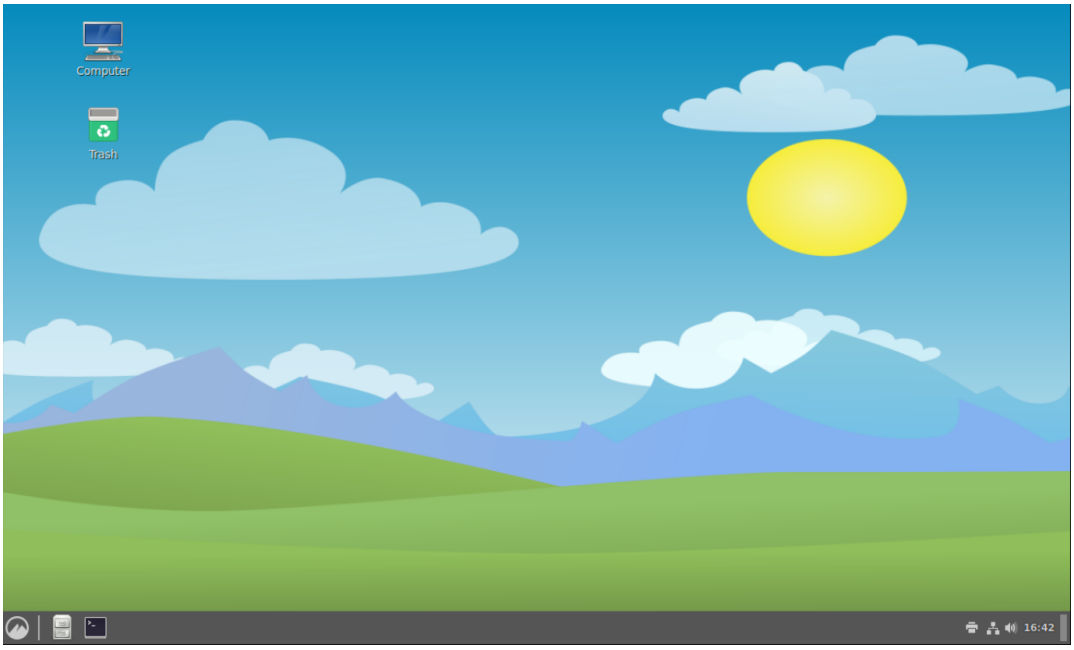
```
exec startxfce4
```

Ayrıca startx yerine doğrudan **startxfce4** komutunu kullanabilirsiniz.

Bununla birlikte elogind gerektiği için elogind etkinleştirmelisiniz.

Cinnamon

Linux mint tarafından geliştirilen kullanıcı dostu bir masaüstü ortamıdır.



Kurulumu

Ymp kullanarak cinnamon yüklemek için aşağıdaki komutu kullanabilirsiniz.

```
$ ymp install @cinnamon
```

Cinnamon masaüstü ortamı gnome çatalıdır ve bazı uygulamaları doğrudan gnome uygulamasıdır. Bunlar masaüstünün parçası olarak gelmediği için kendiniz yüklemelisiniz.

```
$ ymp install gnome-terminal gnome-screenshot polkit-gnome
```

xinitrc ayarları

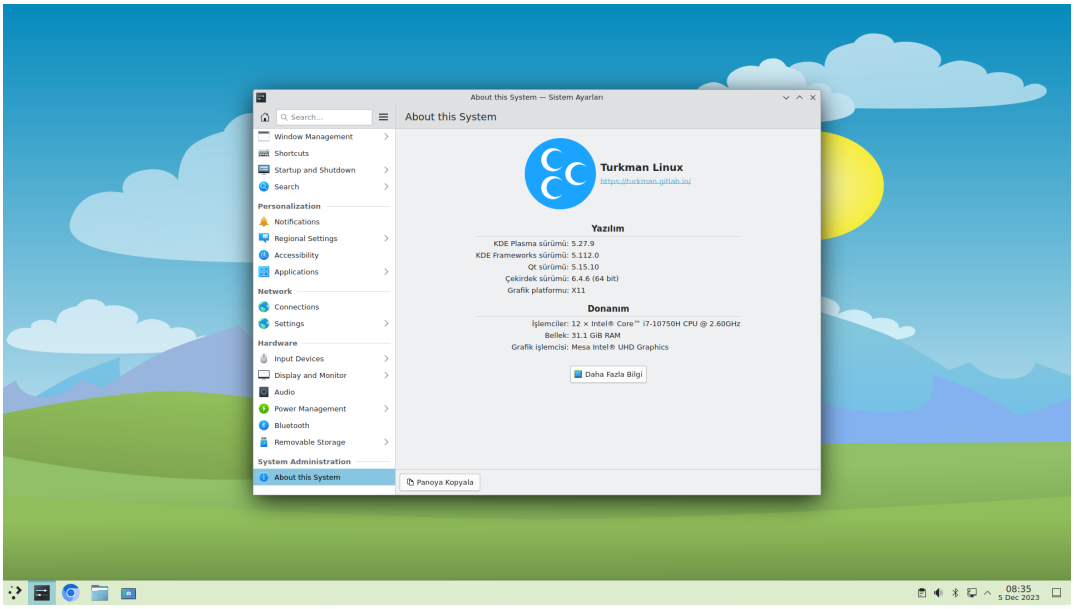
startx komutu çalıştırdığınızda xfce başlatılmasını istiyorsanız **~/.xinitrc** dosyanızın sonuna aşağıdaki gibi ekleme yapmalısınız.

```
exec cinnamon-session-cinnamon
```

Bununla birlikte elogind gerektiği için elogind etkinleştirmelisiniz.

Kde

Kde kolay kullanılan ve oldukça fazla özelliğe sahip masaüstü ortamlarından biridir.



Kurulumu

Ymp kullanarak kde yüklemek için aşağıdaki komutu kullanabilirsiniz.

```
$ ymp install @kde.plasma @kde.frameworks dolphin konsole
```

Ek paketleri yüklemek için ise **@kde.extra** cd **@kde.apps** yüklemelisiniz.

xinitrc ayarları

startx komutu çalıştırdığınızda kde başlatılmasını istiyorsanız **~/.xinitrc** dosyanızın sonuna aşağıdaki gibi ekleme yapmalısınız.

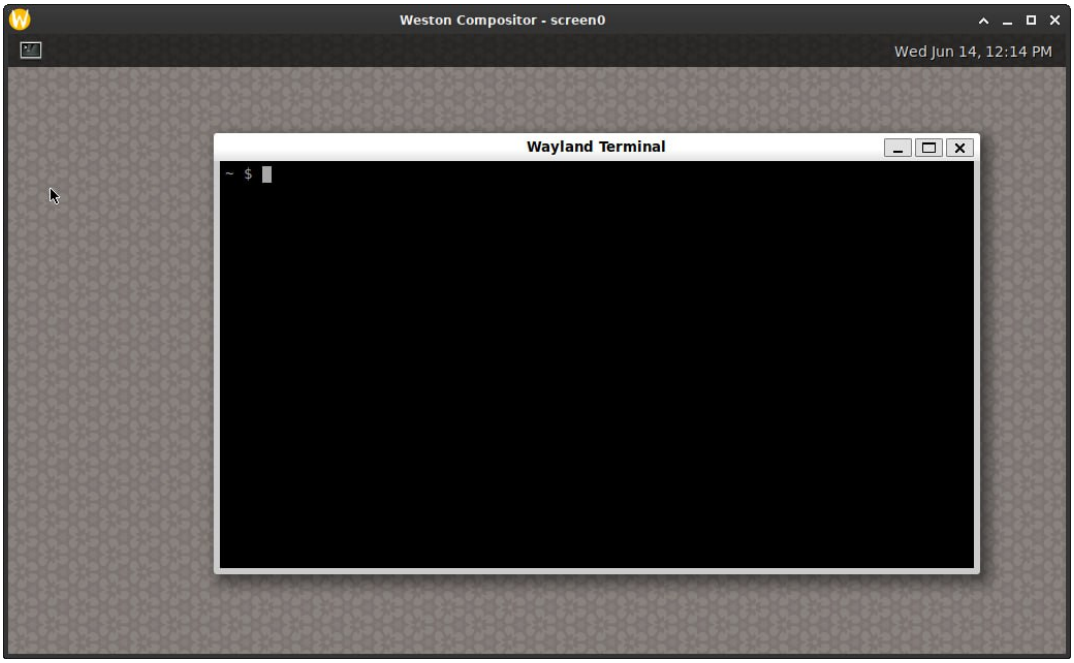
```
exec startplasma-x11
```

Bununla birlikte elogind gerektiği için elogind etkinleştirmelisiniz.

Pencere Yöneticileri

Weston

Weston basit wayland pencere yöneticisidir. Genellikle test amaçlı kullanılır fakat masaüstü ortamı olarak kullanmak da mümkündür.



Yükleme

Ymp kullanarak weston yüklemek için **weston** paketini yüklemelisiniz.

Ardından **seatd** ve **devfs** servislerini etkinleştirmelisiniz.

```
# paketi kuralım
$ ymp install weston
# servisleri etkinleştirmek için
$ rc-update add seatd
$ rc-update add devfs
# servisleri açmak için
$ rc-service devfs start
$ rc-service seatd start
```

Çalıştırma

Çalıştırmak için tty ekranındayken eğer **seatd** servisi açıksa doğrudan **weston** komutu ile oturum açabilirsiniz. Service açmadıysanız **seatd-launch weston** komutu kullanabilirsiniz.

Not: X11 içerisinde weston çalıştırırsanız weston pence modunda çalıştırılacaktır. Bu sayede X11 üzerinde weston uygulamalarını test etebilirsiniz.

Weston'un Parametreleri

Weston'u çalıştırdığınızda, birkaç parametre ile özelleştirebilirsiniz. İşte temel kullanım ve bazı parametreler:

```
$ weston [OPTIONS]
    -B, --backend=BACKEND: Weston'un kullanacağı grafik arka uçunu belirtir. Örneğin, "drm-backend", "wayland-backend", "fbdev-backend" gibi.
    --fullscreen: Weston'u tam ekran modunda başlatır.
    --width=WIDTH, --height=HEIGHT: Weston penceresinin genişliğini ve yüksekliğini belirtir.
    --no-config: Weston'u hiçbir yapılandırma dosyasını kullanmadan başlatır.
    --config=FILE: Belirli bir yapılandırma dosyasını kullanarak Weston'u başlatır.
    --idle-time=SECONDS: Weston'un ne kadar süre boşta kaldıktan sonra oturumu kapatması gerektiğini belirtir.
    --log=LEVEL: Weston'un log seviyesini belirtir (örneğin, "debug", "error", "critical").
```

Örnek bir kullanım:

```
$ weston --backend=drm-backend --fullscreen --width=1920 --height=1080
```

Bu komut, Weston'u DRM (Direct Rendering Manager) arka uç kullanarak tam ekran modunda ve 1920x1080 çözünürlükte başlatacaktır.

Yapılandırma

Weston'un yapılandırma dosyası ~/.config/weston.ini veya /etc/xdg/weston/weston.ini gibi konumlarda bulunabilir. Bu dosyayı özelleştirebilirsiniz.

Örneğin:

```
[core]
modules=desktop-shell.so

[shell]
background-image=/usr/share/backgrounds/gnome/Aqua.jpg
```

Bu örnekte, Weston'un masaüstü kabuğu olarak "desktop-shell" modülünü kullanmasını ve bir arka plan resmi belirtmesini sağlar. Weston Modülleri

Weston modülleri, Weston'un davranışını genişleten eklentilerdir. Örneğin, "desktop-shell" modülü masaüstü yönetimini sağlar. Weston modüllerini weston.ini dosyanızda belirleyebilirsiniz.

sway

Sway wayland pencere yöneticisidir. Sway X11 ortamındaki i3 pencere yöneticisinın benzeridir.



Yükleme

Ymp kullanarak sway yüklemek için **sway** paketini yüklemelisiniz.

Ardından **seatd** ve **devfs** servislerini etkinleştirmelisiniz.

```
# paketi kuralım
$ ymp install sway
# servisleri etkinleştirmek için
```

```
$ rc-update add seatd
$ rc-update add devfs
# servisleri açmak için
$ rc-service devfs start
$ rc-service seatd start
```

Çalıştırma

Çalıştırmak için tty ekranındayken eğer **seatd** servisi açıksa doğrudan **sway** komutu ile oturum açabilirsiniz. Service açmadıysanız **seatd-launch sway** komutu kullanabilirsiniz.

Kullanımı

Temel kısayollar aşağıdaki gibi sıralanabilir:

- *super* + *enter* = yeni terminal aç
- *super* + *ok tuşu* = pencereler arası geçiş
- *super* + *shift* + *ok tuşu* = pencere konumunu ayarla
- *super* + *[1-9]* = masaüstüne geç
- *super* + *shift* + *[1-9]* = pencereyi masaüstüne taşı
- *super* + *Shift* + *Q* = pencereyi kapat
- *super* + *shift* + *E* = sway ortamını kapat

Ekran görüntüsü alma

grim kullanarak ekran görüntüsü alabilirsiniz.

```
# tüm ekranın görüntüsünü almak için
$ grim -o output.png
```

Ekranın bir bölümünü almak isterseniz **slurp** komutu ile birlikte kullanabilirsiniz.

Kısa yol olarak yapılandırma dosyasına şunu ekleyebilirsiniz.

```
# Prtsc tuşu ile parçalı ekran görüntüsü
bindsym Print exec slurp | grim -g -
# super + Prtsc ile tüm ekran görüntüsü
bindsym $mod+Print exec grim
```

Yapılandırma

/etc/sway/config dosyası ana yapılandırma dosyasıdır. Bununla birlikte **~/.config/sway/config** dosyasından kendi kullanıcınıza özel ayarlama yapabilirsiniz.

Hyprland

Hyprland wayland pencere yöneticisidir. Animasyonlara sahip olup düşük kaynak kullanımını hedefler.



Yükleme

Ymp kullanarak sway yüklemek için **hyprland** paketini yüklemelisiniz.

Ardından **seatd** ve **devfs** servislerini etkinleştirmelisiniz.

```
# paketi kuralım
$ ymp install @hyprland
# servisleri etkinleştirmek için
$ rc-update add seatd
$ rc-update add devfs
# servisleri açmak için
$ rc-service devfs start
$ rc-service seatd start
```

Çalıştırma

Çalıştırmak için tty ekranındayken eğer **hyprland** servisi açıksa doğrudan **hyprland** komutu ile oturum açabilirsiniz. Service açmadıysanız **seatd-launch hyprland** komutu kullanabilirsiniz.

Kullanımı

Temel kısayollar aşağıdaki gibi sıralanabilir:

- *super* + *q* = yeni terminal aç
- *super* + *e* = yeni dosya yöneticisi aç
- *super* + *ok tuşu* = pencereler arası geçiş
- *super* + *[1-9]* = masaüstüne geç
- *super* + *j* = pencereleri dik/yan konumlandır.
- *super* + *shift* + *[1-9]* = pencereyi masaüstüne taşı ve o masaüstüne geç
- *super* + *c* = pencereyi kapat
- *super* + *m* = hyprland ortamını kapat

Ekran görüntüsü alma

grim kullanarak ekran görüntüsü alabilirsiniz.

```
# tüm ekranın görüntüsünü almak için
$ grim -o output.png
```

Ekranın bir bölümünü almak isterseniz **slurp** komutu ile birlikte kullanabilirsiniz.

Yapılandırma

Bununla birlikte **~/.config/hyprland/hyprland.conf** dosyasından kendi kullanıcınıza özel ayarlama yapabilirsiniz.

X11

X11 linux dağıtımlarında kullanılan bir grafik ortamıdır. Çoğu masaüstü ortamı ve pencere yöneticisi X11 kullanır.

Elle Başlatma

Öncelikle **X** komutu ile X11 başlatalım.

```
X :0
```

Bu komut boş X ekranı açacaktır.

Not: Bu komutu root ile çalıştırmalısını. Veya kullanıcınız3 **tty** gurubuna eklemelisiniz.

Ardından **DISPLAY** değişkenimizi ayarlayalım.

```
export DISPLAY=:0
```

Daha sonra gui uygulamamızı başlatalım.

```
xterm
```

xinit ile başlatma

Önceki konuda anlatılan şekilde bir masaüstü ortamı başlatmak uzun ve zahmetli olduğu için **xinit** kullanabiliriz.

Öncelikle kullanıcı ev dizinimize **.xinitrc** dosyası açalım ve aşağıdaki gibi içini dolduralım.

```
exec xterm
```

Ardından **startx** komutu ile oturum başlatabilirsiniz.

```
startx
```

Not: Kullanıcınızı **tty** gurubuna eklemelisiniz.

Uzak makinaya pencere yollamak

Öncelikle uzak makinadaki X11 dinleme modunda çalışmalıdır.

Bunun için **/etc/X11/xinit/xserverrc** dosyamızı aşağıdaki gibi ayarlayalım.

```
# -nolisten yerine -listen gelmeli
# -ac eklenmeli.
exec /usr/bin/Xorg -listen tcp -ac "$@"
```

Not: -ac parametresi farklı hostname değerine sahip makinadan ekran açabilmeyi sağlar.

Ardından pencere gönderecek makinada **DISPLAY** değişkenimizi aşağıdaki gibi ayarlamalıyız.

```
# örneğin ip 192.160.0.31 olsun
# örneğin :0 açık olsun
export DISPLAY=192.168.0.31:0
```

Şimdi pencere yollayabiliriz.

```
xterm
```

Önemli Not: Ağ bağlantınız yavaşsa performans sorunları olabilir.

Xephyr

Xephyr X11 içerisinde izole şekilde çalışan başka bir X11 ortamı sağlar. Bu sayede uygulamaları kolayca test edebilirsiniz.

Başlatmak için:

```
Xephyr :1
```

Daha sonra içerisinde bir uygulama açmak için aşağıdaki gibi kullanabiliriz.

```
export DISPLAY=:1
xterm
```

Xvfb

Xvfb herhangi bir ekran olmadan sanal bir X11 ortamı oluşturmayı sağlar. Bu sayede bir uygulamayı ekran kullanmadan test edebilirsiniz.

Kullanımı:

```
xvfb-run xterm
```

Bununla birlikte xvfb oturumu başlatıp kullanabilirsiniz.

```
# Xvfb arkada başlatalım
Xvfb :1 &
# xterm çalıştıralım
export DISPLAY=:1
xterm
```

HDPI

Yüksek çözünürlüklü ve küçük boyutlu ekranlar HDPI (High Dots Per Inch) ekran olarak adlandırılır.

Bu tür ekranlarda varsayılan olan ölçekleme değeri ile kullanım yazıların çok küçük kalmasına sebep olabilir. Bu sebeple ölçekleme gerekebilir.

Dpi hesaplama

Dpi pixel yoğunluğunu ifade eder ve ne kadar büyükse o kadar yüksek kaliteli bir ekrana sahip olduğunuz anlamına gelir. Ekranlar için 96 ve üstü değerler normaldir. Daha düşük değerler ile ekranın kalitesiz olduğunu ifade eder. Dpi hesaplamak için ekranın inç türünden boyutu ve çözünürlük bilgisi gerekmektedir.

Aşağıdaki python kodu ile hesaplanabilir.

```
def calc_dpi(size, height, width):
    dpi = (height**2 + width**2)**0.5 / size
    print(dpi)
# örneğin 1920x1080 ve 23 inç ekran için
# (1920**2 + 1080**2)**0.5 / 23 = 95.77857261227383
# örneğin 2560x1440 ve 27 inç ekran için
# (2540**2 + 1440**2)**0.5 / 27 = 94.9646963498124
```

Yukarıdaki örnek ekran boyutu ve çözünürlükleri yaklaşık 96dpi olduğu için ölçeklemeye gerek yoktur.

Sistem varsayılan dpi değerini 96 olarak alır. Ölçeklendirmeyi bu değere göre yapmalısınız.

Gtk için

Gtk ölçeklemek için **GDK_SCALE** değişkeni ayarlanmalıdır.

```
# %200 ölçekleme için
export GDK_SCALE=2
```

Yukarıdaki tanımı xinitrc dosyanıza yazmanız gerekmektedir.

Not: Sadece tam sayılarda ölçekleme yapılabilmektedir.

Qt için

Qt ölçeklemek için **QT_SCALE_FACTOR** değişkeni ayarlanmalıdır.

```
# %200 ölçeklemek için
export QT_SCALE_FACTOR=2
```

Yukarıdaki tanımı xinitrc dosyanıza yazmanız gerekmektedir.

Xft.dpi ayarı

Ölçekleme desteği bulunmayan araç setlerinde yalnızca yazıları büyütmeyi deneyebilirsiniz. Bu kötü gözükür fakat en azından okunabilir.

Bunun için **~/.Xresources** dosyasına aşağıdaki gibi ekleme yapalım.

```
Xft.dpi: 192
```

Ardından **xrdb -merge ~/.Xresources** komutu ile ayarları uygulayalım.

Not: Gtk ve Qt için yapılan ölçekleme ile aynı anda kullanılmamalıdır.

Diğer Konular

Busybox ile Minimal Dağıtım Oluşturma

Busybox tek bir ikili dosya olarak temel linux komutlarını içerisinde barındıran bir dosyadır. Bu dosya ve kernel olduğu zaman sistemimiz açılıçacak temel komutları kullabileceğimiz bir linux elde etmiş oluruz.

İlk olarak busyboxu çalışma dizinimize kopyalayalım. Busyboxun static olarak derlenmiş olduğundan emin olalım.

```
$ mkdir distro
$ cd distro
$ install /bin/busybox ./busybox
$ ldd ./busybox
-> özdevimli bir çalıştırılabilir değil
```

Ardından initramfs için init dosyamızı aşağıdaki gibi oluşturalım.

```
#!/busybox ash
PATH=/bin
/busybox mkdir /bin
/busybox --install -s /bin
exec /busybox ash
```

initramfs dosyamızı paketleyelim.

```
$ chmod +x init
$ find ./ | cpio -H newc -o > initrd.img
# isterseniz initrd.img sıkıştırabilirsiniz.
$ gzip -9 initrd.img
```

Bu aşamada isterseniz initd.img dosyasını sıkıştırabilirsiniz.

Sıra initrd.img ve kernelin birleştirilmesine geldi. Bunun için aşağıdaki gibi dizin yapısına dosyalarımızı kopyalayalım. vmlinuz dosyamızı kendi sistemimizdeki /boot içinden alabiliriz.

```
iso/vmlinuz
iso/initrd.img
iso/boot/grub/grub.cfg
```

Burada grub.cfg dosyamız bootloader komutlarını içerir. İçerisine aşağıdaki gibi olmalıdır.

```
linux /vmlinuz
initrd /initrd.img
boot
```

Son olarak iso dosyamızı paketleyelim.

```
$ grub-mkrescue iso/ -o distro.iso
```

Minimal sistemimiz hazır. Test etmek için qemu kullanabilirsiniz.

```
$ qemu-system-x86_64 -cdrom distro.iso -m 1G
```

Burada busybox yerine isterseniz static olarak derlenmiş herhangi bir C dosyasını kullanabilirsiniz. Yapmanız gereken init dosyası yerine bu dosyayı kullanmaktır.

LD_PRELOAD

Bir uygulamada bulunan bir fonksiyonu **LD_PRELOAD** yazarak değiştirebiliriz. Bunun için aşağıdaki gibi bir C kodumuz olsun.

```
#include <stdio.h>

int test(char* msg){
    puts(msg);
    return 0;
}

void main(){
    return test("Hello World");
}
```

Bu dosyayı derleyelim.

```
gcc -o main main.c
```

Buradaki değiştirmek istediğimiz test fonksiyonunu aşağıdaki gibi ayrı bir dosyaya yazalım.

```
#include <stdio.h>

int test(char* mgs){
    puts("Hmmm");
    return 1;
}
```

Bu dosyayı **shared** olarak derleyelim.

```
gcc test.c -o test.so -shared
```

main dosyamızı aşağıdaki gibi çağırlırsak değiştirmek istediğimiz fonksiyon çalışır.

```
LD_PRELOAD=$PWD/test.so ./main
```

Eğer değiştirdiğimiz fonksiyona ihtiyacımız varsa aşağıdaki gibi kullanarak elde edebiliriz.

```
#include <stdio.h>
#include <dlfcn.h> /* dlsym için */

int test(char* mgs){
    int (*orig)(char*) = dlsym(RTLD_NEXT, "test");
    int status = orig("HmmmV2");
    return status+2;
}
```

Yukarıdaki gibi yaparak asıl fonksiyona **orig** adı ile erişebiliriz.

Not: Türkmen linuxta güvenlik gereği bu çevreler değişken görmezden gelinir. bunun yerine dosya konumunu **/etc/ld.so.preload** içine yazmalısınız.

Musl libc

Musl hafif ve minimal yapılı bir **libc** alternatifidir.

Kurulum

Türkmen linuxa yüklemek için aşağıdaki komutu kullanabilirsiniz:

```
$ ymp install musl
```

Kaynak koddan derleyip yüklemek için ise öncelikle aşağıdaki gibi bir yol izleyebilirsiniz:

```
# kaynak kodu aşağıdaki gibi derleyip yükleyebiliriz.
$ ./configure --prefix=/usr
$ make
$ make install
```

Not: Eğer sisteme yüklemek istemiyorsanız **--prefix** parametresini kullanıcınıza ait bir dizin ile değiştirmeniz yeterlidir.

Kullanımı

Musl libc kulanarak kaynak kod derlemek için **musl-gcc** komutu kullanılır.

Örneğin basit bir C kodu yazalım ve derleyelim:

```
#include <stdio.h>
int main(){
    puts("Hello World");
}
```

Şimdi bu kodu **musl-gcc** ile derleyelim.

```
$ musl-gcc -o main main.c
```

Meson Cmake veya **Autotools** gibi araçlar ile musl kullanmak isterseniz **CC** değişkeni ayarlamanız gerekmektedir:

```
$ export CC=musl-gcc
```

Şimdi derlenmiş olan koda **musl-ldd** ile bakalım:

```
$ musl-ldd main
    /lib64/ld-musl-x86_64.so.1 (0x7f6a9cfc4000)
    libc.so => /lib64/ld-musl-x86_64.so.1 (0x7f6a9cfc4000)
```

Burada dikkat etmemiz gereken eğer **glibc** tarafından gelen **ldd** komutunu kullanırsak şuna benzer bir hata ile karşılaşırdık:

```
$ ldd main
./main: error while loading shared libraries: /lib64/libc.so: invalid ELF header
```

Çalıştırılması

Musl libc ile derlenmiş bir kaynak kod **interpreter** ve **libc.so** dosyasına ihtiyaç duyar. Bu iki dosya musl libc için aslında aynı dosyadır. Kodu eğer musl libc sistemde yüklü ise doğrudan çalıştırabilirsiniz.

```
./main
```

Eğer sisteminizde kurulu değilse aşağıdaki gibi bir yolu kullanmanız gerekebilir.

```
LD_LIBRARY_PATH=/musl/libc/path /musl/libc/path/lib/ld-musl-x86_64.so.1 ./main
```

Not: musl libc ile derlenmiş bir kod glibc ile derlenmiş kütüphaneleri kullanamaz. Aynı şekilde glibc ile derlenmiş kod musl libc ile derlenmiş kütüphaneleri kullanamaz.

Türkmen Linux Hardened özellikleri

Türkmen linuxu diğer dağıtımlardan ayıran en önemli özelliği **hardened** (sıkılaştırılmış) linux dağıtımı olmasıdır. Bu durum bazı farklılıklara sebep olmaktadır.

Komut listesi gizleme:

/usr/bin ve /bin dizinlerinin izinleri yalnızca çalıştırılabilir şekilde ayarlandığında, sistem üzerinde çalışan uygulamalar bu dizinlerdeki komutların varlığını listeleyemez. Bu durum, terminalde Tab tuşu ile komut tamamlama işlemini kısıtlar.

Bu yöntem, kullanıcıların yalnızca belirtilen komutları çalıştırmasına ve diğer komutların görünürlüğünü azaltarak güvenlik sağlar. Özellikle hassas sistemlerde, komutların gizlenmesi, yetkisiz erişim riskini minimize etmenin etkili bir yolu olarak öne çıkar.

Canlı sistem:

Türkmen Linux, canlı modda çalıştırıldığında Zram oluşturur ve bu alanı rastgele bir parola ile LUKS (Linux Unified Key Setup) protokolüyle şifreler. Bu sayede canlı sistem üzerinde yapılan değişiklikler şifreli şekilde tutulduğu için fazladan koruma sağlar.

Çekirdek:

Türkmen linux çekirdek yapılandırması varsayılan olarak hardened olarak sağlanır.

Bununla birlikte **sysctl.conf** yapılandırmaları önyüklü olarak gelmektedir.

Varsayılan olarak bazı çekirdek modülleri (uvcvideo gibi) kapalı olarak gelir.

Initramfs:

Türkmen linux initramfs sistemi olarak **mkinitrd** kullanır. Bu initramfs sistemi hook çalıştırmak için cgroup kullanır ve sistemi başlatmadan önce kendisi dışında kalan tüm süreçleri öldürür. Bu sayede initramfs içerisinden kendisi dışında başka bir şeyin çalıştırılmasına engel olur.

Açılış sırasında **OEM** var mı diye kontrol edilir ve tespit edilmesi durumunda kullanıcıya uyarı mesajı verilir.

Sistem seviyesi önlemler:

/tmp dizini **noexec** olarak bağlanır. Bu sayede herhangi bir kullanıcının buraya bir dosya koyup çalıştırmasının önüne geçilir.

Kök dizin **nosuid** olarak bağlanır. Suid ihtiyacı olan dosyalar **/etc/suid.d** dizini altından okunur ve açılışta sadece o dosyalara suid yetkisi verilir.

/sys/class/dmi dizine boş dizin bağlanır. Bu sayede çalışan uygulamaların hangi donanımı kullandığınızı görmesi engellenir.

/proc dizini **hidepid=2** olarak bağlanır. Tüm çalışan süreçleri görmek için kullanıcının **proc** gurubunda yer alması gerekmektedir.

Pipewire sistem servisi olarak çalışmaktadır. Bu sayede /dev/snd dizinine erişime ihtiyaç duymadan pipewire kullanılabilmektedir.

su komutu olarak suidless-su kullanılır. https://codeberg.org/turkmen-linux/suidless-su Bu sayede suid biti gerekmeden çalışır.

Diğer

Udisks2 parolasız disk bağlama özelliği kapatılmıştır. Disk bağlamak için yetkili kullanıcı gerekmektedir.

Sık karşılaşılabilen problemler

Bu başlık altında sıkça karşılaşılabilen hatalar ve olası çözümleri anlatılmaktadır.

docbook-xsl / docbook-xml hataları

Eğer docbook ile ilgili **add command failed** hatası alıyorsanız. aşağıdaki yolu izleyin.

1- /etc/xml/catalog dosyasını silin

```
rm -f /etc/xml/catalog
```

2- docbook-xml ve docbook-xsl paketlerinin sysconf yapılandırmasını silin.

```
$ rm -f /var/lib/ymp/sysconf/docbook-xsl/postinstall.done
$ rm -f /var/lib/ymp/sysconf/docbook-xml/postinstall.done
```

3- sysconf modellerini sırası ile elle tetikleyin.

```
$ bash -e /etc/sysconf.d/docbook-xml
$ bash -e /etc/sysconf.d/docbook-xsl
```

Segmentation fault hatası

Bu hata genellikle bir C kütüphanesinin veya kodunun hatasından kaynaklanır. Hatanın nereden kaynaklandığını bulmak için **gdb** kullanabiliriz. Bunun için öncelikle hata veren kodun **-g3** parametresi ile derlenmesi gerekir. Örneğin:

```
$ export CFLAGS="-g3" # C için
$ export CXXFLAGS="-g3" # C++ için
# Bu kısım değişiklik gösterebilir.
$ configure --prefix=/usr
$ make
```

```
$ make install
# basit bir C kodunu gcc ile derleyeceksek
$ gcc -g3 main.c -o main
```

Ardından kodumuzu gdb yardımı ile çalıştıralım.

Örneğin aşağıdaki gibi hatalı yazılmış bir C kodumuz olsun:

```
int main(){
    char* test; // pointer tanımladık.
    test[1] = 'c'; // mevcut olmayan bir yere yazmaya çalıştık.
    return 0;
}
```

```
$ gdb ./main
...
(gdb) run
...
Program received signal SIGSEGV, Segmentation fault.
main () at main.c:3
3           test[1] = 'c';
```

Bu hatanın nereden geldiğini öğrenmek için **backtrace** kullanabiliriz:

```
(gdb) backtrace
#0  main () at main.c:3
```

Hatanın bulunduğu satıra giderek kaynak koddaki sorunu bulup düzeltebilirsiniz.

aclocal-1.x bulunamadı hatası

Bu hata genellikte eski sürüm automake kullanılarak yapılandırılmış projelerde meydana gelir ve aşağıdakine gibi bir hatadır.

```
build-aux/missing' aclocal-1.17 -I m4
.../build-aux/missing: line 85: aclocal-1.17: command not found
WARNING: 'aclocal-1.17' is missing on your system.
         You should only need it if you modified 'acinclude.m4' or
         'configure.ac' or m4 files included by 'configure.ac'.
make: *** [Makefile:2427: aclocal.m4] Error 127
```

Bu hatayı çözmek için *autoreconf -vfi* komutunu kullanabilirsiniz. Alternatif olarak şu komut ile eski sürüme uyumlu hale getirebilirsiniz.

```
ln -s aclocal /usr/local/bin/aclocal-1.17
ln -s automake /usr/local/bin/automake-1.17
```

Programlama

Bash dersi

Bu yazıda bash betiği yazmayı hızlıca anlatacağım. Bu yazıda karıştırılmaması için girdilerin olduğu satırlar <- ile çıktıların olduğu satırlar -> ile işaretlenmiştir.

Açıklama satırı ve dosya başlangıcı

Açıklamalar **#** ifadesiden başlayıp satır sonuna kadar devam eder. Dosyanın ilk satırına **#!/bin/bash** eklememiz gerekmektedir. Bash betikleri genellikle **.sh** uzantılı olur. Bash betikleri girintilemeye duyarlı değildir. Bash betiği yazarken girintileme için 4 boşluk veya tek tab kullanmanızı öneririm.

Bash betiklerinde alt satıra geçmek yerine **;** kullanabiliriz. Bu sayede kaynak kod daha düzenli tutulabilir.

```
#!/bin/bash
#Bu bir açıklama satırıdır.
```

Bash betiklerini çalıştırmak için öncelikle çalıştırılabilir yapmalı ve sonrasında aşağıdaki gibi çalıştırılmalıdır.

```
chmod +x ders1.sh
./ders1.sh
```

: komutu hiçbir iş yapmayan komuttur. Bu komutu açıklama niyetine kullanabilirsiniz. **true** komutu ile aynı anlama gelmektedir.

Çoklu açımlama satırı için aşağıdaki gibi bir ifade kullanabilirsiniz.

```
: "
Bu bir açıklama satıdırıdır.
Bu da diğer açıklama satırıdır.
Bu da sonunca açıklama satırıdır.
"
```

Ekrana yazı yazalım

Ekrana yazı yazmak için **echo** ifadesi kulanılır.

```
echo Merhaba dünya
-> Merhaba dünya
```

Ekrana özel karakterleri yazmak için **-e** parametresi kullanmamız gerekmektedir.

```
echo -e "Merhaba\ndünya"
-> Merhaba
-> dünya
```

Ekrana renkli yazı da yazdırabiliriz. Bunun için **\033[x;..;ym** ifadesini kullanırız. Burada **x** ve **y** özellik belirtir. Örneğin:

```
# Mavi renkli kalın merhaba ile normal dünya yazdırır.
echo -e "\033[34;1mMerhaba\033[0m Dünya"
```

Aşağıda tablo halinde özellik numarası ve anlamları verilmiştir.

Özellik numarası ve anlamları

Özellik	Anlamı	Özellik	Anlamı	Özellik	Anlamı
0	Sıfırla	30	Siyah yazı	40	Siyah arka plan
1	Aydınlık	31	Kırmızı yazı	41	Kırmızı arka plan
2	Sönük	32	Yeşil yazı	42	Yeşil arka plan
3	İtalik	33	Sarı yazı	43	Sarı arka plan
4	Altı çizili	34	Mavi yazı	44	Mavi arka plan
5	Yanıp sönen	35	Magenta yazı	45	Magenta arkaplan
6	Yanıp sönen	36	Turkuaz yazı	46	Turkuaz arka plan
7	Ters çevirilmiş	37	Beyaz yazı	47	Beyaz arka plan
8	Gizli	39	Varsayılan yazı	49	Varsayılan arkaplan

Çift tırnak (") içine yazılmış yazılardaki değişkenler işlenirken tek tırnak (') içindekiler işlenmez. Örneğin:

```
var=12
echo "$var"
echo '$var'
-> 12
-> $var
```

Parametreler

Bir bash betiği çalıştırılırken verilen parametreleri **$** ifadesinden sonra gelen sayı ile kullanabiliriz. **$#** bize kaç tane parametre olduğunu verir. **$@** ifadesi ile de parametrelerin toplamını elde edebiliriz.

```
echo "$1 - $# - $@"

./ders1.sh merhaba dünya
-> merhaba - 2 - merhaba dünya
```

Değişkenler ve Sabitler

Değişkenler ve sabitler programımızın içerisinde kullanılan verilerdir. Değişkenler tanımlandıktan sonra değiştirilebilirken sabitler tanımlandıktan sonra değiştirilemez.

Değişkenler sayı ile başlayamaz, Türkçe karakter içeremez ve */*[[($* gibi özel karakterleri içeremez.

Normal Değişkenler aşağıdaki gibi tanımlanır.

```
sayı=23
yazi="merhaba"
```

+= ifadesi var olan değişkene ekleme yapmak için kullanılır. Değişkenin türünü belirlemeden tanımlamışsak yazı olarak ele alır.

```
typeset -i a # Değişkeni sayı olarak belirttik.
a=1 ; b=1
a+=1 ; b+=1
echo "$a $b"
-> 2 11
```

Çevresel değişkenler tüm alt programlarda da geçerlidir. Çevresel değişken tanımlamak için başına **export** ifadesi yerleştirilir.

```
export sayi=23
export yazi="merhaba"
```

Sabitler daha sonradan değeri değiştirilemeyen verilerdir. Sabit tanımlamak için başına **decrale -r** ifadesi yerleştirilir.

```
declare -r yazi="merhaba"
declade -r sayi=23
```

Değişkenler ve sabitler kullanılırken **${}** işareti içine alınırlar veya başına **$** işareti gelir. Bu doküman boyunca ilk kullanım biçimi üzerinden gideceğim.

```
deneme="abc123"
echo ${deneme}
-> abc123
```

Sayı ve yazı türünden değişkenler farklıdır. sayıyı yazıya çevirmek için **"** işaretleri arasına alabiliriz. Birden fazla yazıyı toplamak için yan yana yazmamız yeterlidir.

```
sayi=11
yazi="karpuz"
echo "${sayi}${karpuz} limon"
-> 11karpuz limon
```

Sayı değişkenleri üzerinde matematiksel işlem yapmak için aşağıdaki ifade kullanılır. *(+-*/ işlemleri için geçerlidir.)*

```
sayi=12
sayi=$((${sayi}/2))
echo ${sayi}
-> 6
```

Bununla birlikte matematiksel işlemler için şunlar da kullanılabilir.

```
expr 3 + 5 # her piri arasında boşluk gerekli
-> 8
echo 6-1 | bc -l # Burada -l virgüllü sayılar için kullanılır.
```

```
-> 5
python3 -c "print(10/2)"
-> 5.0
```

Değişkenlere aşağıdaki tabloda belirttiğim gibi müdahale edilebilir. Karakter sayısı 0'dan başlar. Negatif değerler sondan saymaya başlar.

Değişkene müdahale (var="Merhaba")

İfade	Anlamı	Eşleniği
${var%aba}	Sondaki ifadeyi sil	Merh
${var#Mer}	Baştaki ifadeyi sil	haba
${var:1:4}	Baştan 1. 4. karakterler arası	erha
${var::4}	Baştan 4. karaktere kadar	Merha
${var:4}	Baştan 4. karakterden sonrası	aba
${var/erh/abc}	erh yerine abc koy	Mabcaba
${var,,}	hepsini küçük harf yap	merhaba
${var^^}	hepsini büyük harf yap	MERHABA
${var:+abc}	var tanımlıysı abc dördürür.	abc

Diziler

Diziler birden çok eleman içeren değişkenlerdir. Bash betiklerinde diziler aşağıdaki gibi tanımların ve kullanılır.

```
dizi=(muz elma limon armut)
echo ${dizi[1]}
-> elma
echo ${#dizi[@]}
-> 4
echo ${dizi[@]:2:4}
-> limon armut
dizi+=(kiraz)
echo ${dizi[-1]}
-> kiraz
```

Diziler eleman indisleri ile kullanmanın yanında şu şekilde de tanımlanabilir.

```
declare -A dizi
dizi=([kirmizi]=elma [sari]=muz [yesil]=limon [turuncu]=portakal)
for isim in ${!dizi[@]} ; do
    echo -n "$isim "
done
echo
-> turuncu yesil sari kirmizi
for isim in ${dizi[@]} ; do
    echo -n "$isim "
done
```

```
echo
-> portakal limon muz elma
echo ${dizi[kirmizi]}
-> elma
```

Klavyeden değer alma

Klavyeden değer almak için **read** komutu kullanılır. Alınan değer değişken olarak tanımlanır.

```
read deger
<- merhaba
echo $deger
-> merhaba
```

Koşullar

Koşullar **if** ile **fi** ile biter. Koşul ifadesi sonrası **then** kullanılır. ilk koşul sağlanmıyorsa **elif** ifadesi ile ikinci koşul sorgulanabilir. Eğer hiçbir koşul sağlanmıyorsa **else** ifadesi içerisindeki eylem gerçekleştirilir.

```
if ifade ; then
    eylem
elif ifade ; then
    eylem
else
    eylem
fi
```

Koşul ifadeleri kısmında çalıştırılan komut 0 döndürüyorsa doğru döndürmüyorsa yalnış olarak değerlendirilir. **[[** veya **[** ile büyük-küçük-eşit kıyaslaması, dosya veya dizin varlığı vb. gibi sorgulamalar yapılabilir. Bu yazıda **[[** kullanılacaktır.

```
read veri
if [[ ${veri} -lt 10 ]] ; then
    echo "Veri 10'dan küçük"
else
    echo "Veri 10'dan büyük veya 10a eşit"
fi

<- 9
-> Veri 10'dan küçük
<- 15
-> Veri 10'dan büyük veya 10a eşit
```

[[yerleşiği ile ilgili başlıca ifadeleri ve kullanımlarını aşağıda tablo olarak ifade ettim. **[** bir komutken **[[** bir yerleşiktir. **[** ayrı bir süreç olarak çalıştırılır. Bu yüzden **[[** kullanmanızı tavsiye ederim.

[[ifadeleri ve kullanımları

İfade	Anlamı	Kullanım şekli
-lt	küçüktür	[[${a} -lt 5]]

-gt	büyüktür	[[${a} -gt 5]]
-eq	eşittir	[[${a} -eq 5]]
-le	küçük eşittir	[[${a} -le 5]]
-ge	büyük eşittir	[[${a} -ge 5]]
-f	dosyadır	[[-f /etc/os-release]]
-d	dizindir	[[-d /etc]]
-e	vardır (dosya veya dizindir)	[[-e /bin/bash]]
-L	sembolik bağdır	[[-L /lib]]
-n	uzunluğu 0 değildir	[[-n ${a}]]
-z	uzunluğu 0dır	[[-z ${a}]]
!	ifadenin tersini alır.	[[! .... veya ! [[....
>	alfabeti olarak büyüktür	[["portakal" > "elma"]]
<	alfabetik olarak küçüktür	[["elma" < "limon"]]
==	alfabetik eşittir	[["nane" == "nane"]]
!=	alfabetik eşit değildir	[["name" != "limon"]]
=~	regex kuralına göre eşittir	[["elma1" =~ ^[a-z]l.*[1]$]]
||	mantıksal veya bağlacı	[[.... || ....]] veya [[....]] || [[....]]
&&	mantıksal ve bağlacı	[[.... && ....]] veya [[....]] && [[....]]

true komutu her zaman doğru **false** komutu ile her zaman yanlış çıkış verir.

Bazı basit koşul ifadeleri için if ifadesi yerine aşağıdaki gibi kullanım yapılabilir.

```
[[ 12 -eq ${a} ]] && echo "12ye eşit." || echo "12ye eşit değil"
#bunun ile aynı anlama gelir:
if [[ 12 -eq ${a} ]] ; then
    echo "12ye eşit"
else
    echo "12ye eşit değil"
fi
```

case yapısı

case yapısı case ile başlar değerden sonra gelen **in** ile devam eder ve koşullardan sonra gelen **esac** ile tamamlanır. case yapısı sayesinde if elif else ile yazmamız gereken uzun ifadeleri kısaltabiliriz.

```
case deger in
    elma | kiraz)
        echo "meyve"
        ;;
    patates | soğan)
```

```
        echo "sebze"
        ;;
    balık)
        echo "hayvan"
    *)
        echo "hiçbiri"
        ;;
esac
# Şununla aynıdır:
if [[ "${deger}" == "elma" || "${deger}" == "kiraz" ]] ; then
    echo "meyve"
elif [[ "${deger}" == "patates" || "${deger}" == "soğan" ]] ; then
    echo "sebze"
elif [[ "${değer}" == "balık" ]] ; then
    echo "hayvan"
else
    echo "hiçbiri"
fi
```

Döngüler

Döngülerde **while** ifadesi sonrası koşul gelir. **do** ile devam eder ve eylemden sonra **done** ifadesi ile biter. Döngülerde ifade doğru olduğu sürece eylem sürekli olarak tekrar eder.

```
while ifade ; do
    eylem
done
```

Örneğin 1den 10a kadar sayıları ekrana yan yana yazdıralım. Eğer echo komutumuzda **-n** parametresi verilirse alt satıra geçmeden yazmaya devam eder.

```
i=1
while [[ ${i} -le 10 ]] ; do
    echo -n "$i " # sayıyı yazıya çevirip sonuna yanına boşluk koyduk
    i=$((${i}+1)) # sayıya 1 ekledik
done
echo # en son alt satıra geçmesi için
-> 1 2 3 4 5 6 7 8 9 10
```

for ifadesinde değişken adından sonra **in** kullanılır daha sonra dizi yer alır. diziden sonra **do** ve bitişte de **done** kullanılır.

```
for degisken in dizi ; do
    eylem
done
```

Ayrı örneğin for ile yapılmış hali

```
for i in 1 2 3 4 5 6 7 8 9 10 ; do
    echo -n "${i} "
done
echo
-> 1 2 3 4 5 6 7 8 9 10
```

Ayrıca uzun uzun 1den 10a kadar yazmak yerine şu şekilde de yapabiliyoruz.

```
for i in {1..10} ; do
    echo -n "${i} "
done
echo
-> 1 2 3 4 5 6 7 8 9 10
```

Buradaki özel kullanımları aşağıda tablo halinde belirttim.

küme parantezli ifadeler ve anlamları

İfade	Anlamı	eşleniği
{1..5}	aralık belirtir	1 2 3 4 5
{1..7..2}	adımlı aralık belirtir	1 3 5 7
{a,ve}li	kurala uygun küme belirtir	ali veli

Fonksiyonlar

Fonksiyonlar alt programları oluşturur ve çağırıldığında işlerini yaptıktan sonra tekrar ana programdan devam edilmesini sağlar. Bir fonksiyonu aşağıdaki gibi tanımlayabiliriz.

```
isim(){
    eylem
    return sonuç
}
# veya
function isim(){
    eylem
    return sonuç
}
```

Burada **return** ifadesi kullanılmadığı durumlarda 0 döndürülür. return ifadesinden sonra fonksiyon tamamlanır ve ana programdan devam edilir.

Bu yazı boyunca ilkini tercih edeceğiz.

Fonksionlar sıradan komutlar gibi parametre alabilirler ve ana programa ait sabit ve değişkenleri kullanabilirler.

```
sayi=12
topla(){
    echo $((${sayi}+$1))
    return 0
    echo "Bu satır çalışmaz"
}
topla 1
-> 13
```

local ifadesi sadece fonksionun içinde tanımlanan fonksion bitiminde silinen değişkenler için kullanılır.

Fonksionların çıkış turumlarını koşul ifadesi yerine kullanabiliriz.

```
read sayi
teksayi(){
    local i=$(($1+1)) # sayıya 1 ekledik ve yerel hale getirdik.
    return $((${i}%2))  # sayının 2 ile bölümünden kalanı döndürdük
}
if teksayi ${sayi} ; then
    echo "tek sayıdır"
else
    echo "çift sayıdır"
fi

<- 12
-> çift sayıdır
<- 5
-> tek sayıdır
```

Bir fonksionun çıktısını değişkene **$(isim)** ifadesi yadımı ile atayabiliriz. Aynı durum komutlar için de geçerlidir.

```
yaz(){
    echo "Merhaba"
}
echo "$(yaz) dünya"
-> Merhaba dünya
```

Tanımlı bir fonksionu silmek için **unset -f** ifadesini kullanmamız gereklidir.

```
yaz(){
    echo "Merhaba"
}
unset -f yaz
echo "$(yaz) dünya"
-> bash: yaz: komut yok
-> dünya
```

Burada dikkat ederseniz olmayan fonksionu çalıştırmaya çalıştığımız için hata mesajı verdi fakat çalışmaya devam etti. Eğer herhangi bir hata durumunda betiğin durmasını istiyorsak **set -e** bu durumun tam tersi için **set +e** ifadesini kullanmalıyız.

```
echo "satır 1"
acho "satır 2" # yanlış yazılan satır fakat devam edecek
echo "satır 3"
set -e
acho "satır 4" # yanlış yazılan satır çalışmayı durduracak
echo "satır 5" # bu satır çalışmayacak
-> satır 1
-> bash: acho: komut yok
-> satır 3
-> bash: acho: komut yok
```

Dosya işlemleri

Bash betiklerinde **stdout stderr** ve **stdin** olmak üzere 2 çıktı ve 1 girdi bulunur. Ekrana stderr ve stdout beraber yazılır.

dosya ifadeleri ve anlamları

İfade	Türü	Anlamı
stdin	Girdi	Klavyeden girilen değerler.
stdout	Çıktı	Sıradan çıktılardır.
stderr	Çıktı	Hata çıktılarıdır.

cat komutu ile dosya içeriğini ekrana yazdırabiliriz. Dosya içeriğini **$(cat dosya.txt)** kullanarak değişkene atabiliriz.

dosya.txt içeriğinin aşağıdaki gibi olduğunu varsayalım.

```
Merhaba dünya
Selam dünya
sayı:123
```

Ayağıdaki örnekle dosya içeriğini önce değişkene atayıp sonra değişkeni ekrana yazdırdık.

```
icerik=$(cat ./dosya.txt)
echo "${icerik}"
-> Merhaba dünya
-> Selam dünya
-> sayı:123
```

grep "sözcük" dosya.txt ile dosya içerisinde sözcük gezen satırları filitreleyebiliriz. Eğer grep komutuna **-v** paraketresi eklersek sadece içermeyenleri filitreler. Eğer filitrelemede hiçbir satır bulunmuyorsa yanlış döner.

```
grep "dünya" dosya.txt
-> Merhaba dünya
-> Selam dünya
grep -v "dünya" dosya.txt
-> sayi:123
```

Aşağıdaki tabloda bazı dosya işlemi ifadeleri ve anlamları verilmiştir.

dosya ifadeleri ve anlamları

İfade	Anlamı	Kullanım şekli
>	çıktıyı dosyaya yönlendir (stdout)	echo "Merhaba dünya" > dosya.txt
2>	çıktıyı dosyaya yönlendir (stderr)	ls /olmayan/dizin 2> dosya.txt
>>	çıktıyı dosyaya ekle	echo -n "Merhaba" > dosya.txt && echo "dünya" >> dosya.txt
&>	çıktıyı yönlendir (stdout ve stderr)	echo "$(cat /olmayan/dosya) deneme" &> dosya.txt

Ayrıca dosyadan veri girişleri için de aşağıda örnekler verilmiştir:

```
# <<EOF:
# EOF ifadesi gelene kadar olan kısmı girdi olarak kullanır:
cat > dosya.txt <<EOF
Merhaba
dünya
EOF
# < dosya.txt
# Bir dosyayı girdi olarak kullanır:
while read line ; do
    echo ${line:2:5}
done < dosya.txt
```

/dev/null içine atılan çıktılar yok edilir. **/dev/stderr** içine atılan çıktılar ise hata çıktısı olur.

Boru hattı

Bash betiklerinde **stdin** yerine bir önceki komutun çıktısını kullanmak için boru hattı açabiliriz. Boru hattı açmak için iki komutun arasına **|** işareti koyulur. Boru hattında soldan sağa doğru çıktı akışı vardır. Boru hattından sadece **stdout** çıktısı geçmektedir. Eğer **stderr** çıktısını da boru hattından geçirmek istiyorsanız **|&** kullanmalısınız.

```
topla(){
    read sayi1
    read sayi2
    echo $((${sayi1}+${sayi2}))
}
topla
<- 12
<- 25
-> 37
sayiyaz(){
    echo 12
    echo 25
}
 sayiyaz | topla
-> 37
```

Kod bloğu

{ ile **}** arasına yazılan kodlar birer kod bloğudur. Kod blokları fonksionların aksine argument almazlar ve bir isme sahip değillerdir. Kod blokları tanımlandığı yerde çalıştırılırlar. Kod bloğuna boru hattı ile veri girişi ve çıkışı yapılabilir.

```
cikart(){
    read sayi1
    read sayi2
    echo $((${sayi1}-${sayi2}))
}
cikart
<- 25
<- 12
-> 13
{
    echo 25
```

```
    echo 12
} | cikart
-> 13
# veya kısaca şu şekilde de yapılabilir.
{ echo 25 ; echo 12 ; } | cikart
-> 13
```

select komutu

select kullanarak basit menü oluşturabiliriz.

```
select deger in ali veli 49 59 ; do
    echo $REPLY # seçilen sayıyı verir
    echo $deger # seçilen elemanı verir
    break
done

-> 1) ali
-> 2) veli
-> 3) 49
-> 4) 59
-> #?
<- 1
-> 1
-> ali
```

Bu örnekde **REPLY** değişkeni seçtiğimiz sayıyı **deger** değişkeni ise seçtiğimiz elemanı ifade eder. **select** komutu sürekli olarak döngü halinde çalışır. Döngüden çıkmak için **break** kullandık.

Birden çok dosya ile çalışmak

Bash betikleri içerisinde diğer bash betiği dosyasını kullanmak için **source** yada **.** ifadeleri kullanılır. Diğer betik eklendiği zaman içerisinde tanımlanmış olan değişkenler ve fonksionlar kullanılabilir olur.

Örneğin deneme.sh dosyamızın içeriği aşağıdaki gibi olsun:

```
mesaj="Selam"
merhaba(){
    echo ${mesaj}
}
echo "deneme yüklendi"
```

Asıl betiğimizin içeriği de aşağıdaki gibi olsun.

```
source deneme.sh # deneme.sh dosyası çalıştırılır.
merhaba
-> deneme yüklendi
-> Selam
```

Ayrıca bir komutun çıktısını da betiğe eklemek mümkündür. Bunun için **<(komut)** ifadesi kullanılır. Aşağıda bununla ilgili bir örnek verilmiştir.

```
source <(curl https://gitlab.com/sulincix/outher/-/raw/gh-pages/deneme.sh) # Örnekteki adrese takılmayın :D
merhaba
merhaba2
echo ${sayi}
-> Merhaba dünya
-> 50
-> 100
```

exec komutu

exec komutu betiğin bundan sonraki bölümünü çalıştırmak yerine hedefteki komut ile değiştirilmesini sağlar. **exec** ile çalıştırılmış olan komut tamamlandığında betik tamamlanmış olur.

```
echo $$ # pid değeri yazdırır
bash -c 'echo $$' # yeni süreç oluşturduğu için pid değeri farklıdır.
exec bash -c 'echo $$' # mevcut komut ile değiştirildiği için pid değeri değişmez
echo "hmm" # Bu kısım çalıştırılamaz.
-> 5755
-> 5756
-> 5755
```

exec komutunu doğrudan terminalde çalıştırırsanız ve komut tamamlanırsa terminaldeki süreç kapanacağı için terminal doğal olarak kapanacaktır.

exec komutunu kullanarak yönlendirmeler yapabilirsiniz.

```
exec > log.txt # bütün çıktıları log.txt içine yazdırır.
echo "merhaba" # ekrana değil dosyaya yazılır.
exec < komutlar.txt # komutlar.txt dosyasındakiler girdi olarak kullanılır.
```

fd kavramı

bash programında birden çok **fd** kullanılabilir. var olan fdlere ulaşmak için **/proc/$$/fd/** konumuna bakabiliriz. 0 stdin 1 stdout 2 stderr olarak çalışır.

Not: $$ mevcut sürecin pid değerini verir.

```
exec 3> log.txt # yazmak için boş bir 3 numaralı fd açmak için.
echo "deneme" >&3
exec 3>& - # açık olan 3 nuramalı fd kapatmak için.
exec 2>&1 # stderr içine atılanı stdout içine aktarır.
exec 4< input.txt # okumak için 4 numaralı fd açmak için.
echo "hmm" > input.txt # girdi dosyamıza yazı yazalım.
read line <&4 # 3 nuramalı fd içinden değir okur.
exec 4<&- # 4 numaralı fd kapatmak için.
```

Hata ayıklama

bash komutuna farklı parametreler vererek kolayca script'inizi derleyebilirsiniz. Örneğin **-n** parametresi kodu çalıştırmayıp sadece hata kontrolü yapacaktır, **-v** komutları çalıştırmadan yazdıracak, **-x** ise işlem bittikten sonra kodları yazdıracaktır.

```
bash -n script_adi.sh
bash -v script_adi.sh
bash -x script_adi.sh
```

C Dersi

Bu derste C programlama dersi anlatılacaktır. Bu dersin düzgünce anlaşılabilmesi için temel düzey gnu/linux bilmeniz gerekmektedir.

Derleme işlemi

C derlemeli bir programlama dilidir. Yani yazılan kodun derleneek bilgisayarın anlayacağı hale getirilmesi gerekmektedir. Derleme işlemini **gcc** veya **clang** kullanarak yapabiliriz.

```
# koddan .o dosyası üretelim
$ gcc -c main.c
# .o dosyasından derlenmiş dosya üretelim.
$ gcc -o main main.o
# kodu çalıştıralım
$ ./main
-> Hello World
```

Yukarıdaki örnekte öncelikle **.o** uzantılı object dosyamızı ürettik. Bu dosya kodun derlenmiş fakat henüz kullanıma hazır hale getirilmemiş halidir. Bu sebeple **.o** dosyalarını linkleme işleminden geçirerek son halini almasını sağlamalıyız.

Not: derleyicimiz **.o** üretmeden de doğrudan derleme yapabilir.

```
$ gcc -o main main.c
```

Açıklama satırı

C kodlarında 3 farklı yolla girintileme yapılabilir.

1. **//** kullanarak satırın geri kalanını açıklama satırı yapabiliriz.

```
// bu bir açıklama satırıdır.
```

2. **/*** ile başlayıp ***/** ile biten alanlar açıklamadır.

```
/* Bu
   bir
   açıklama
   satırıdır */
```

3. **#if 0** ile başlayan satırdan **#endif** satırına kadar olan kısım açıklama satırıdır.

```
#if 0
bu bir
açıklama
satırıdır
#endif
```

Girintileme

C programlama dilinde blocklar **{}** karakterleri ile belirtilir. Kodun okunalı olması için girintilenmesi gereklidir fakat şart değildir. Girintileme için 4 boşluk veya 1 tab kullanabilirsiniz.

Bir block aşağdaki gibi bir yapıya sahiptir.

```
aaaa (bbbb) {
        cccc;
  ddd;
}
```

Not: Her satırın sonunda ; işareti bulunmalıdır.

İlk program

C programları çalıştırıldığında **main** fonsiyonu çalıştırılır. Aşağıda örnek main fonksiyonu bulunmaktadır.

```
int main(int argc, char** argv) {
    return 0;
}
```

- **int main** kısmında int döndürülecek değer türü main adıdır.
- **int argc** parametre sayısını belirtir.
- **char **argv** parametre listesini belirtir.
- **return 0** komutu 0 ile çıkış yapar.

Burada **main** fonksiyonunu türünün bir önemi yoktur. **void** olarak da tanımlanabilir. Ayrıca kullanmayacaksak arguman tanımlamaya da gerek yoktur. Kısaca Şu şekilde de yazabilirdik.

```
void main(){}
```

Ekrana yazı yazma

Öncelikle **stdio.h** kütüphanesine ihtiyacimiz olduğu için onu eklemeliyiz. Ardından **printf** fonksiyonu ile ekrana yazı yazabiliriz.

printf fonksiyonunun 1. parametresi yazdirma şablonunu diğerleri ise yazdırılacak verileri belirtir.

```
#include <stdio.h>

int main(int argc, char** argv) {
    printf("%s\n", "Merhaba Dünya!");
    return 0;
}
```

- **include** ile belirttiğimiz dosyalar sistemde **/usr/include** içerisinde bulunur.
- **printf** fonksiyonundaki **%s** yazılar için, **%c** karakterler için, **%d** sayılar için kullanılır.

Değişkenler

C dilinde değişkenler aşağıdaki gibi tanımlanır.

```
...
int sayi = 12;
```

```
char* yazi = "test";
char karakter = 'c';
float sayi2 = 12.42;
...
```

Bununla birlikte **#define** kullanarak derlemeden önce koddaki alanların karşılığı ile değiştirilmesini sağlayabilirsiniz. Bu şekilde tanımlanan değerler derlemeden önce yerine yazıldığı için değişken olarak işlem görmezler.

```
#define sayi 12
...
printf("%d\n",sayi);
...
```

Diziler

Diziler iki şekilde tanımlanabilir.

1. Pointer kullanarak tanımlanabilir. Bu konunun detaylarına ilerleyen kısımda değinilecektir. Bu şekilde tanımlanan dizilerde başta uzunluk belirtilmek zorunda değildir.

```
int *dizi = {12, 22, 31};
```

2. Uzunluk belirterek tanımlanabilir. Bu şekilde tanımlanan dizilerin uzunluğu sabittir.

```
int dizi[3] = {12, 22, 31};
```

C dilinde string kavramı bulunmaz. Onun yerine karakter dizileri kullanılır.

```
char *txt = "deneme123";
```

Dizinin bir elemanına erişmek için aşağıdaki gibi bir yol kullanılır.

```
int *dizi = {12, 22, 31};
int c = dizi[1]; // dizinin 2. elemanı
```

Not: Dizi indisleri 0dan başlar.

Bir dizinin uzunluğunu dizinin bellekteki boyutunu birim boyutuna bölerek buluruz. Bunun için **sizeof** fonksiyonu kullanılır.

```
int *dizi = {11, 22, 31};
int l = sizeof(dizi) / sizeof(int);
```

Klavyeden değer alma

Klavyeden değer almak için **scanf** kullanılır. İlk parameter şablonu diğerleri ise değişkenlerin bellek adresini belirtir.

```
int sayi;
scanf("%d\n", &sayi);
```

Not: Bu şekilde değer alma yaptığımızda formata uygun olmayan şekilde değer girilebilir. Eğer böyle bir durum oluşursa değişken **NULL** olarak atanır. yani değeri bulunmaz. Buda kodun işleyişinde soruna yol açabilir. Bu yüzden değişkeni kullanmadan ince **NULL** olup olmadığını kontrol etmelisiniz.

Koşullar

Koşullar için **if** bloğu kullanılır. Block içindeki ifade **0** veya **NULL** olursa koşul sağlanmaz. Bu durumda varse **else** bloğu çalıştırılır.

```
if (koşul1) {
    block 1
} else if (koşul2) {
  block 2
} else {
  block 3
}
```

Örnek olarak girilen sayının çift olup olmadığını yazan uygulama yazalım.

```
#include <stdio.h>

int main(int argc, char** argv) {
    int sayi;
    scanf("%d",&sayi);
    if (sayi == NULL) {
        printf("%s\n", "Geçersiz sayı girdiniz.");
    } else if(sayi % 2) {
        printf("%d tektir.\n", sayi);
    } else {
        printf("%d çifttir.\n", sayi);
    }
    return 0;
}
```

Burada **%** operatörü 2 ile bölümden kalanı bulmaya yarar. Sayı tek ise 1 değilse 0 sonucu elde edilir. Bu sayede tek sayılar için koşul sağlanır çift sayılar için sağlanmaz.

Tek satırdan oluşan koşullarda **{}** kullanmaya gerek yoktur.

```
if (i < 32)
  printf("%s\n","32den küçüktür");
```

Koşul ifadeleri aşağıdaki gibi listelenebilir.

Koşul işleyicileri

ifade	anlamı	örnek
>	büyüktür	121 > 12
<	küçüktür	12 < 121
==	birbirine eşittir	121 == 121
!	karşıtlık bildirir.	!(12 > 121)

&&	logic and	"fg" == "aa" && 121 > 12
||	logic or	"fg" == "aa" || 121 > 12
!=	eşit değildir	"fg" != "aa"
>=	büyük eşittir	121 >= 121
<=	küçük eşittir	12 <= 12

Switch - Case

Bir sayıya karşılık bir işlem yapmak için **switch - case** yapısı kullanılır.

```
      switch(sayi) {
        1:
          // sayı 1se burası çalışır.
          // break olmadığı için alttan devam eder.
        2:
          // sayı 1 veya 2 ise burası çalışır.
          break;
        3:
          // sayı 3 ise burası çalışır.
        default:
          // sayı eşleşmezse burası çalışır.
}
```

Döngüler

Döngüler koşullara benzer fakat döngülerde koşul sağlanmayana kadar block içi tekrarlanır. Döngü oluşturmak için **while** kullanılır.

```
int i=10;
while(i<0){
    printf("%d\n", i);
    i--;
}
```

Yukarıdaki örnekte 10dan 0a kadar geri sayan örnek verilmiştir. En son i değişkeni 0 olduğunda koşul sağlanmadığı için döngü sonlanır.

Aynı işlemi **for** ifadesi ile de yapabiliriz.

```
for(int i=10;i<0;i--){
    printf("%d\n", i);
}
```

Burada for içerisinde 3 bölüm bulunur. İlkinde değer atanır. İkincinde koşul yer alır. Üçüncüsünde değişkene yapılacak işlem belirtilir.

Döngülerde **continue** kullanarak döngünün tamamlanması beklenmeden başa dönülür. **break** kullanarak döngüden çıkılır.

```
      int sayi = 10
      while(1) {
          printf("%d\n",sayi);
    if(sayi < 0) {
```

```
            break;
        }
        sayi--;
        continue;
        printf("%s\n","Bu satıra gelinmez.");
}
```

Yukarıdaki örnekte döngü koşulu sürekli olarak devam etmeye neden olur. Sayımız 0dan küçükse döngü **break** kullanarak sonlandırılır. Döngü içinde **continue** kısmına gelindiğinde başa dönüldüğü için bir alttaki satır çalıştırılmaz.

goto

C dilinde kodun içerisindeki bir yere etiket tanımlanıp **goto** ile bu etikete gidilebilir.

```
yaz:
printf("%s\n", "Hello World");
goto yaz;
```

Yukarıdaki örnekte sürekli olarak yazı yazdırılır. Bunun sebebi her seferinde **yaz** etiketine gidilmesidir.

Bundan faydalanarak döngü oluşturulabilir.

```
int i = 10;
islem:
if(i < 0){
    printf("%d\n",i);
    i--;
    goto islem;
}
```

Burada koşul bloğunun en sonunda tekrar başa dönmesi için **goto** kullandık.

Fonksiyonlar

C dilinde bir fonksiyon aşağıdaki gibi tanımlanır.

```
int yazdir(char* yazi){
    if(yazi != NULL){
        printf("%s\n",yazi);
        return 0;
    }
return 1;
}
```

Yukarıdaki fonksiyon verilen değişken değere sahipse ekrana yazdırıp 0 döndürür. Eğer değeri yoksa 1 döndürür.

Basit işlemler için **#define** ile de fonksiyon tanımlanabilir. Bu şekilde tanımlanan fonksiyonlar derleme öncesi yerine yazılarak çalışır.

```
#define topla(A,B) A+B

int main(int argc, char** argv){
```

```
    int sayi = topla(3, 5);
    return 0;
}
```

Fonksiyonlar yazılma sırasına göre kullanılabilirler. Bu yüzden fonksiyonlar henüz tanımlı değilse kullanılamazlar. Bu durumun üstesinden gelmek için **header** tanımlaması yapılır.

```
void yaz();
int main(){
    yaz();
    return 0;
}
void yaz(){
    printf("%s\n","Hello World");
}
```

Header tanımlamaları kütüphane yazarken de kullanılır. Bunun için bu tanımlamaları **.h** uzantılı dosyalara yazmanız gereklidir. Bu dosyayı **include** kullanarak eklemeliyiz.

yaz.h dosyası

```
void yaz();
```

main.c dosyası

```
#include "yaz.h"
#include <stdio.h>

int main(){
    yaz();
    return 0;
}

void yaz(){
    printf("%s\n","Hello World");
}
```

Not: include ifadesinde **<>** içine aldığımız dosyalar **/usr/include "**" içine aldığımız ise mevcut dizinde aranır.

Pointer ve Address kavramı

Pointerlar bir değişkenin bellekte bulunduğu yeri belirtir. ve * işareti ile belirtir. Örneğin aşağıda bir metin pointer olarak tanımlansın ve 2 birim kaydırılsın.

```
char* msg = "abcde";
printf("%s\n", msg + sizeof(char)*2 );
```

Bura 2 char uzunluğu kadar pointer kaydırıldığı için ekrana ilk iki karakteri silinerek yazdırılmıştır.

Adres ise bir değişkenin bellek adresini ifade eder. **&** işareti ile belirtilir. Örneğin rastgele bir değişken oluşturup adresini ekrana yazalım.

```
int i = 0;
printf("%p\n" &i);
```

Konunun daha iyi anlaşılması için bir değişken oluşturup adresini bir pointera kopyalayalım. ve sonra değişkenimizi değiştirelim.

```
int i = 0; // değişken tanımladık.
int *k = &i; // adresini kopyaladık.
int l = i; // değeri kopyaladık.
i = 1; // değişkeni değiştirdik.
printf("%d %d\n", i, *k, l);
```

Bu örnekte ilk iki değer de değişir fakat üçüncüsü değişmez. Bunun sebebi ikinci be birinci değişkenlerin adresi aynıyken üçüncü değişkenin adresi farklıdır.

Bir fonksiyon tanımlarken pointer olarak arguman aldırıp bu değerde değişiklik yapabilir. Buna örnek kullanım olarak **scanf** fonksiyonu verilebilir.

```
#include <stdio.h>
void topla(int* sonuc, int sayi1, int sayi2){
    *sonuc = sayi1 + sayi2;
}
void main(){
    int i;
    topla(&i, 12, 22);
    printf("%d\n",i);
}
```

Burada fonksiyona değişkenin adresi girilir. Fonksiyon bu adrese toplamı yazar. Daha sonra değişkenimizi kullanabilirsiniz.

Fonksiyonun kendisini de pointer olarak kullanmak mümkündür. Bunun için aşağıdaki gibi bir yapı kullanılabilir.

```
int topla(int i, int j){
    return i + j;
}

void main(){
    int (*topla_func)(int, int) = topla;
    topla_func(3, 5);
}
```

Ayrıca **typedef** yapısı ile de fonksiyon pointerları oluşturulabilir. Bu konunun detaylarına ilerleyen kısımlarda yer verilmiştir.

```
typedef int (*topla_func)(int, int);
int topla(int i, int j){
    return i + j;
}

void main(){
    topla_func topla_fn = topla;
```

```
    topla_fn(3, 5);
}
```

Dinamik bellek yönetimi

Dinamik bellek yönetimi için **malloc**, **realloc**, **calloc**, **free** fonksiyonları kullanılır. Bu fonksiyonlar **stdlib.h** ile sağlanmaktadır.

malloc fonksiyonu belirtilen boyut kadar boş alanı **void*** olarak tahsis eder.

```
// 10 elemanlı sayı dizisi oluşturmak için.
int *sayilar = (int*) malloc(10 * sizeof(int));
// şununla aynı anlama gelir.
int sayilar[10];
```

calloc fonksiyonu malloc ile benzerdir fakat istenen block boyutunu da belirterek kullanılır.

```
// 10 elemanlı sayı dizisi oluşturmak için
int *sayilar = (int*) calloc(10, sizeof(int));
// şununla aynı anlama gelir
int *sayilar = (int*) malloc(10 * sizeof(int));
```

realloc bir değişkenin yeniden boyutlandırılmasını sağlar.

```
// 5 elemanlı dizi tanımlayalım.
int sayilar[5];
// boyutu 10 yapalım
sayilar = (int*) realloc(sayilar, 10*sizeof(int));
```

free fonksiyonu değişkeni bellekten siler.

```
// malloc ile bir alan tanımlayalım.
void* alan = malloc(100);
// bu alanı silelim.
free(alan);
```

Konunun daha iyi anlaşılması için 2 stringi toplayan fonksiyon yazalım.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
char* add(char *s1, char *s2){
    int ss = strlen(s1); // ilk arguman uzunluğu
    int sx = strlen(s2); // ikinci arguman uzunluğu
    char* s3 = (char*)malloc(ss+sx*sizeof(char)); // uzunluklar toplamı kadar alan ayır.
    for(int i=0;s1[i];i++) // ilkinin tüm elemanlarını kopyala
        s3[i] = s1[i];
    for(int i=0;s2[i];i++) // ikincinin tüm elemanlarını kopyala
        s3[i+ss] = s2[i];
    s3[ss+sx]='\0'; // stringler '\0' ile sonlanır
    return s3;
}
```

```
void main(){
    char *new_str = add( "hello", "world");
    printf("%s\n", new_str);
}
```

Bellek yönetimi iki ana kısımda olur:

1. Stack (Yığın): Derleyicinin otomatik olarak yönettiği bellek alanı.
2. Heap (Öbek): Programcı tarafından yönetilen bellek alanı.

Bu iki bellek alanını kullanarak bir program yazalım.

```
#include <stdio.h>
void main() {
    int a; // stack'de yer ayırır ve değer atar.
    int *b = malloc(sizeof(int)); // öbek'ten bellek ayırır. Değer atanmamış.
    int c;
    b = &a; // pointer artık işaret ettiği adres değişmiş oluyor.
    scanf("%d",b);
    printf("Değer: %d\n",*b);
}
```

Stack belleği derleyici tarafından otomatik olarak serbest bırakılır. Heap bellekleri için de serbest bırakmak gereklidir.

Struct

Structure yapıları bellekte belli bir değişken topluluğu oluşturup kullanabilmek için kullanılır. Bu yapılar sayesinde kendi veri türlerinizi tanımlayabilirsiniz.

```
struct test {
    int num;
    char* name;
};

void main(){
    struct test t1;
    t1.num = 12;
    t1.name = "hello";
}
```

Veri türü adına alias tanımlamak için **typedef** kullanılabilir. Bu sayede değişken tanımlar gibi tanımlama yapmak mümkündür.

```
typedef struct Test {
    int num;
    char* name;
} test;

void main(){
    test t1;
    t1.num = 12;
    t1.name = "hello";
}
```

typedef kullanarak struct dışında değişken türü tanımlamak da mümkündür.

```
typedef char* my_string;

void main(){
    my_string str = "Hello World";
}
```

C programlama dili nesne yönelimli bir dil değildir. Bu yüzden sınıf kavramı bulunmaz. Fakat **struct** kullanarak benzer işler yapılabilir. Bunun için fonksiyon pointeri tanımlayıp struct yapımıza ekleyelim. Bir init fonksiyonu kullanarak nesnemizi oluşturalım.

```
// nesne struct yapısı tanımladık
typedef struct Test {
    // nesne fonksiyonunu tanımladık.
    void (*yazdir)(char*);
    int num;
} test;

// nesne fonksiyon işlevin tanımladık
void test_yazdir(char* msg){
    printf("%s\n",msg);
}

// nesneyi oluşturan fonsiyonu tanımladık.
test test_init(){
    test t1;
    t1.num = 12;
    t1.yazdir = test_yazdir;
    return t1;
}

void main(){
    test obj = test_init();
    obj.yazdir("Hello World");
}
```

Kütüphane dosyası oluşturma

Kütüphaneler ana kaynak kodun kullandığı yardımcı kodları barındırır. Bu sayede her uygulama için tek tek aynı şeyleri yazmak yerine tek bir kütüphaneden yararlanılabilir.

GNU/Linux ortamında kütüphaneler **.so** uzantılıdır ve **/lib** ve **/usr/lib** dizinlerinde bulunur.

Not: Ek kütüphane dizinlerini **/etc/ld.so.conf** ve **/etc/lo.so.conf.d/*** dosyalarında belirlenir. Bunula birlikte **LD_LIBRARY_PATH** çevresel değişkeni ile kütüphane dizini tanımı yapılabilir.

Bir dosyanın bağımlı olduğu kütüphaneleri **ldd** komutu ile görüntüleyebiliriz.

```
$ ldd /bin/bash
    /lib/ld-musl-x86_64.so.1 (0x7fd299f6d000)
    libreadline.so.8 => /usr/lib/libreadline.so.8 (0x7fd299e5e000)
    libc.musl-x86_64.so.1 => /lib/ld-musl-x86_64.so.1 (0x7fd299f6d000)
    libncursesw.so.6 => /usr/lib/libncursesw.so.6 (0x7fd299e0a000)
```

Kendi kütüphanemizi olşturmak için kaynak kodumuzu **-shared** parametresi ile derlememiz gerekmektedir. Bunu için örneğin aşağıdaki gibi bir kaynak kodumuz olsun.

```
int topla (int a, int b) {
    return a+b;
}
```

Bu kodu derleyelim.

```
$ gcc -c test.c
$ gcc -o libtest.so test.o -shared
```

Şimdi de bu kütüphaneyi kullanabilmek için **test.h** dosyamızı oluşturalım.

```
int topla (int a, int b);
```

Son olarak kütüphaneyi kullanan kodumuzu yazalım.

```
#include <test.h>
void main(){
    int sayi = topla(3, 5);
}
```

Dikkat ettiyseniz **include** kullanırken **""** işareti yerine **<>** kullandık. Bunun sebebi kütüphanelerin kaynak koddan bağımsız olacak şekilde tasarlanmasıdır. Header dosyamızın **/usr/include** içinde ve kutuphanemizin de **/usr/lib** içinde olduğunu varsayarak kodladık.

Kütüphanemizin **kutuphane** adındaki bir dizinde bulunduğunu düşünelim ve aşağıdaki gibi derlemeyi tamamlayalım.

```
$ gcc -c main.c -I ./kutuphane
$ gcc -o main main.o -L ./kutuphane -ltest
```

Kodu kütüphaneyi sisteme yüklemeden derleyebilmemiz için derleyicimize **-I** parametresi eklenir. Bu parametre header aradığı dizinlere belirtilen dizini de ekler. Benzer şekilde derlemenin **linkleme** aşamasında **-l** parametresi ile hangi kütüphanelere ihtiyaç duyulduğu belirtilir. **-L** parametresi ile kütüphanenin aranacağı dizinler listesine belirtilen dizin eklenir.

Gördüğünüz gibi bu parametreler sisteme göre değişiklik gösterebilmektedir. Bu karmaşanın önüne geçebilmek için **pkg-config** kullanılır. Bu dosyada belirtilen değerler kütüphane ile beraber gelmekte olup derlemeye nelerin ekleneceğini belirtir.

Örnek olarak aşağıdaki gibi kullanabiliriz.

```
# derleme parametreleri
$ pkg-config --cflags readline
  -DNCURSES_WIDECHAR
# linkleme parametreleri
$ pkg-config --libs readline
  -lreadline
```

Kaynak kodu derlerken aşağıdaki gibi kullanılabilir.

```
$ gcc -c main.c `pkg-config --cflags readline`
$ gcc -o main main.o `pkg-config --libs readline`
```

pkg-config dosyaları **.pc** uzantılıdır ve **/usr/lib/pkgconfig** içinde bulunur. **pkg-config** dosyaları aşağıdaki formata benzer şekilde yazılır.

```
prefix=/usr
includedir=${prefix}/include

Name: Test
Description: Test library
Version: 1.0
Requires: readline
Cflags: -I{includedir}/test
Libs: -ltest -L{libdir}/test
```

Yukarıdaki örnekte **/usr/include/test/** içerisindeki header dosyamızı ve **/usr/lib/test/** içindeki kütüphane dosyamızı sorunsuzca kullanarak derleme yapabilik.

C++ kodunu C ile kullanma

Aşağıdaki gibi bir C++ kodumuz olsun.

```
class test {
private:
    double value;

public:
    test(double l) {
        value = l;
    }
}
```

Bu kodu C ile kullanmak için öncelikle aşağıdaki gibi header dosyamızı oluşturalım.

```
#ifdef __cplusplus
extern "C" {
#endif

typedef void* test; // class tanımı

test test_new(double l); // class oluşturmak için
void test_set_value(test* ctx, double l); // class içindeki değeri ayarlamak için
double test_get_value(test* ctx); // class içindeki değeri okumak için

#ifdef __cplusplus
}
#endif
```

Şimdi de bir C++ kodu yazalım ve yukarıdaki fonksiyonları oluşturalım.

```
#include <test>
extern "C" {
    test test_new(double l){
```

```
        Test* t = new Test(l);
        return (test) t;
    }

    void test_set_value(test* ctx, double l){
        Test* t = (Test*)ctx;
        t->value = l;
    }
    double test_get_value(test* ctx){
        Test* t = (Test*)ctx;
        return t->value;
    }
}
```

Şimdi bu kodu kütüphane haline getirelim.

```
g++ -o libtest.so test.cpp -I. -shared
```

Şimdi örnek C kodumuz aşağıdaki gibi olabilir.

```
#include <test.h>
int main(){
    test *t = test_new(22);
    test_set_value(t,12);
    int l = test_get_value(t);
    return 0;
}
```

Bu kodu aşağıdaki gibi derleyebiliriz.

```
gcc main.c -o main -ltest -L. -I.
```

Yada kütüphane olşturmaya gerek kalmadan da derleyebiliriz.

```
g++ test.cpp -c -o test.o
gcc -c main.c -o main.o
gcc tast.o main.o -o main -lstdc++
```

Burada libstdc++ C++ kodunun bağımlılığıdır.

Docker kullanımı

Bu yazıda sizlere docker kullanımı anlatılacaktır.

Docker kurulumu

Docker kurmak için debian tabanlı dağıtımlarda **docker.io** paketini kurmalısınız.

```
$ apt install docker.io --no-install-recommends
```

Eğer debian dışında bir dağıtım kullanıyorsanız kendi sitesi üzerinden indirmeyi veya kaynak koddan derlemeyi deneyebilirsiniz.

```
# docker binarylerini indirmek için https://download.docker.com/linux/static/stable/
# indirdikten sonra arşivden çıkatrıp containerd ve dockerd çalıştıralım. (root kullanarak)
$ tar -xf docker-xxx.tgz
$ cd docker
$ export PATH=$PATH:$(pwd) # path içine mevcut dizini de ekledik
# containerd ve dockerd çalıştıralım. Bunlar daemon olduğu için sürekli arkada çalışması gerekmektedir.
$ containerd
$ dockerd
```

Rootless docker kurmak için

```
$ curl -fsSL https://get.docker.com/rootless | bash
```

Docker çalışıyor mu diye kontrol etmek için **docker info** komutunu kullanabiliriz.

```
$ docker info
```

Rootless olmayan docker root kullanıcısı ile çalıştığı için root kullanmadan docker çalışmayacaktır. Bunun için /run/docker.sock dosyamızın aitliğini bir guruba verip kullanıcıyı da o guruba alabiliriz. Bu işlem güvenlik sorunlarına sebep olabilir. Detaylı bilgi için: https://docs.docker.com/engine/security/#docker-daemon-attack-surface

```
# Bu aşama docker gurubu mevcut olmayanlar için geçerlidir.
$ groupadd docker
$ chgrp docker /run/docker.sock # bu işlem her yeniden başlatmada gerekebilir.
# Bu aşama tüm dağıtımlarda gereklidir.
$ usermod -aG docker username # oturumu kapatıp açmak gerekebilir.
```

Docker çalışma mantığı

Docker linux çekirdeği üzerinde oluşturulmuş **container** yapısı üzerinde çalışır. Her container kendisine özel ayrılmış alanda kısıtlı kaynaklar ile çalışır. Bu sayede herhangi bir performans kaybı olmadan ana sistemden bağımsız şekilde istenilen uygulamalar çalıştırılabilir.

Docker containerlarını üretmek için öncelikle imaj gerekmektedir. İmajlara **dockerhub** üzerinden ulaşabiliriz. Öncelikle imaj kurulur. daha sonra bu imajdan container türetilir. İş bittikten sonra container kolaylıkla yok edilebilir. Veya containerların imajı alınarak yeni imaja dönüştürülebilir.

```
kernel (linux)
  |
system (gentoo)
  |
docker ----------------
  |       |    |     |
alpine debian arch ubuntu
```

İmajlar

Docker imajı indirmek için **docker pull** komutu kullanılır. **name:tag** şeklinde belirtilir. eğer tag belirtilmemişse latest olarak değerlendirilir.

```
$ docker pull alpine:latest
```

Mevcut imajları listelemek için **docker images** kullanılır.

```
$ docker images
REPOSITORY   TAG       IMAGE ID       CREATED       SIZE
debian       latest    43d28810c1b4   8 days ago    124MB
alpine       latest    9c6f07244728   6 weeks ago   5.54MB
```

Burada dikkat edilmesi gereken konu her imajın bir **ID** değeri bulunmasıdır. İmajlar ile ilgili işlemler yapılırken imajın adı veya bu id değeri kullanılır.

İmaj silmek için **docker rmi** kullanılır.

```
$ docker rmi alpine
# veya şu da kullanılabilir.
$ docker rmi 9c6f07244728
```

İmajımızı bir dosyaya kaydetmek için **docker save** kullanırız.

```
$ docker save debian:latest -o /home/backup/debian.tar
```

kaydedilmiş bir dosyadan imaj yüklemek için ise **docker load** kullanılız.

```
$ docker load -i /home/backup/debian.tar
```

tarball dosyasından docker imajı oluşturmak için **docker import** kullanabiliriz.

```
$ docker import rootfs.tar custom:new
# veya bir dizinden üretebiliriz
$ tar -c -C rootfs . | docker import - custom:new2
```

Docker imajını tarball olarak çıkartmak için ise **docker export** kullanabiliriz.

```
$ docker export debian:stable > /home/user/debian-stable.tar
# veya şu şekilde de kullanılabilir
$ docker export -o /home/user/debian-stable.tar debian:stable
```

Containerlar

Containerlar içerisinde uygulama çalıştırdığımız alanlardır. imajlardan türetilirler. bir container üretmek için **docker run** komutu kullanılır. Bu komut aldığı parametreler ile containerın özelliklerini ayarlar.

```
#  docker run <seçenekler> <imajın idsi veya ismi> <çalıştırılacak komut>
$ docker run -it -d -p 8000:80 --name deneme 43d28810c1b4 /bin/bash
# -i stdin okumasına izin verir
# -d komutu arkada çalıştır
# -t pseudo-tty olarak çalıştırır. -it olarak kullanıp shell çalıştırabiliriz.
# -p port yönlendirmesi yapar.
# --name container ismi ayarlar. Belirtilmemişse rastgele bir isim alır.
```

Eğer container çalıştıktan sonra silinmesini istiyorsanız **--rm** parametresi ekleyebiliriz. Bu sayede işlem bitimi otomatik olarak silinir.

```
$ docker run --rm alpine echo hello world
```

İşlem başlatmayıp sadece container oluşturmak istiyorsanız **docker create** kullanabilirsiniz.

```
$ docker create --name deneme2 debian
```

Container oluştururkenki seçenekler için **docker run --help** veya **docker create --help** yapabilirsiniz.

Çalışan containerları listelemek için **docker ps** kullanılır. **-a** parametresi eklenirse tüm containerlar listelenir. **-q** parametresi ile sadece id değerleri yazdırılır.

```
$ docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED          STATUS                      PORTS     NAMES
e1e2983bfa34   debian    "bash"    8 seconds ago    Exited (0) 5 seconds ago              test
b91e04ab5dcc   debian    "bash"    23 seconds ago   Up 22 seconds                         deneme
```

Çalışan bir containera bağlanmak için **docker attach** kullanılır.

```
# ctrl-k kısayolu ile bağlantı kesilmesi için ek parametre ekleyelim.
$ docker attach b91e04ab5dcc --detach-keys="ctrl-k"
```

Çalışan bir container **docker kill** kullanılarak kapatılabilir. kapatılmış bir container docker start kullanılarak tekrar başlatılabilir.

```
$ docker kill b91e04ab5dcc
$ docker ps -q | grep b91e04ab5dcc # çıktı boşsa container çalışmıyor demektir
$ docker start b91e04ab5dcc
```

Container ile işimiz bittiğinde silmek için **docker rm** kullanılır. Silme işleminden önce kapatmamız gerekir. Eğer zorla kapatılmasını isterseniz **-f** parametresi ekleyebiliriz.

```
$ docker rm b91e04ab5dcc
Error response from daemon: You cannot remove a running container ...
$ docker rm -f b91e04ab5dcc
# Aşağıdaki komutla tüm containerları silebiliriz.
$ docker rm -f $(docker ps -a -q)
```

Çalışmayan tüm containerların silinmesi için **docker container prune** kullanılabilir.

```
$ docker container prune
WARNING! This will remove all stopped containers.
Are you sure you want to continue? [y/N] y
Total reclaimed space: 0B
```

Çalışan containerlar ile ilgili kullanım istatistiklerine ulaşmak için **docker stats** kullanılır. **docker top** ise container içinde çalışan süreçler ile ilgili bilgi almaya yarar.

Container ile ilgili bilgi almak için **docker inspect** kullanılır.

```
$ docker stats
CONTAINER ID   NAME            CPU %     MEM USAGE / LIMIT   MEM %     NET I/O       BLOCK I/O   PIDS
40f84cb8e4e0   deneme2         0.00%     808KiB / 31.15GiB   0.00%     1.87kB / 0B   0B / 0B     1
$ docker top 40f84cb8e4e0
UID            PID             PPID      C                   STIME     TTY           TIME        CMD
root           7432            7396      0                   10:42     pts/0         00:00:00    bash
$ docker inspect 40f84cb8e4e0
```

```
...
  "Id": "40f84cb8e4e0...",
  "Created": "2022-09-21T07:42:18.337126911Z",
...
```

Çalışan bir container içerisinde bir komut çalıştırmak için **docker exec** kullanılır.

```
$ docker exec -it 40f84cb8e4e0 /bin/bash
```

Containerları duraklatıp devam ettirmek için **docker pause** ve **docker unpause** kullanılır.

```
$ docker pause 40f84cb8e4e0
$ docker unpause 40f84cb8e4e0
```

Mevcut containerdan imaj elde etmek için **docker commit** kullanabiliriz.

```
$ docker commit 40f84cb8e4e0 builder:1.0
```

Uzak sunucuda çalışmak

DOCKER_HOST çevresel değişkenini ayarlayarak ssh üzerinden uzaktaki bir makinadaki container ve imajları yönetebilirsiniz.

```
$ export DOCKER_HOST=ssh://user@server
$ docker info
```

Bağlantı için ssh anahtarınızı sunucuya atmış olmanız gerekmektedir. Bunun için **ssh-copy-id** kullanabilirsiniz veya anahtarınızı **~/.ssh/authorized_keys** içerisine yazmalısınız.

```
$ ssh-copy-id user@server
user@server's password:
```

Volume kavramı

Docker üzerinde birden çok container ile çalıştığımızı farz edelim. Bu containerlar birbirleri ile dosya alışverişi yapmak isteyebilirler. Örneğin bir tanesi web server olarak çalışırken diğeri web serverda bulunan dosyaları farklı bir amaç için kullanabilir.

Bu gibi durumlar için **volume** bulunur. Volume container tarafından kullanılabilen depolama alanlarıdır. Volume oluşturmak için **docker volume create** kullanılır.

Volume diskte **/var/lib/docker/volumes/** içerisinde depolanır.

```
$ docker volume create data
```

Var olan **volume** listesi için **docker volume ls** kullanılır.

```
$ docker volume ls
DRIVER    VOLUME NAME
local     data
```

Bir **volume** silmek için **docker volume rm** kullanılır. Silmeden önce bu alanı kullanan containerları kapatmalısınız.

```
$ docker volume rm data
```

Bir container başlatılırken ona volume eklemek için **--mount** parametresi eklenir.

```
$ docker run -d --name webserver --mount source=data,target=/var/www/http/ nginx:latest
```

Bağlanacak dizine yazılmasını istemiyorsak **readonly** eklemeliyiz.

```
docker run --mount source=data,target=/app,readonly test321 alpine
```

Container içine bir dizine tmpfs bağlamak için **type** belirtilir.

```
$ docker run --mount type=tmpfs,target=/app/temp/ --name apptest debian
# Şu şekilde de kullanılabilir.
$ docker run --tmpfs /app/temp/ --name apptest debian
```

Ayrıca volume yerine ana sistemdeki bir dizini de bağlayabiliriz.

```
docker run --mount type=bind,source=/home/shared,target=/shared --name test123 alpine
```

Dizinleri aşağıdaki gibi de bağlayabiliriz.

```
# yazılmasını istemiyorsanız ro istiyorsanız rw
# Hiçbir şey eklemezseniz rw kabul edilir.
docker run -v /mnt:/mnt:ro -v /shared:/shared:rw test456 alpine
```

Dockerfile

Dockerfile docker kullanarak belli işleri gerçekleştirmeye yarayan bir talimat dosyasıdır. Bu talimatların sonucunda yeni bir imaj dosyası oluşturulur. Örneğin aşağıda bir Dockerfile dosyası verilmiştir.

```
FROM alpine
RUN echo hello world
```

Bir Dockerfile dosyası aşağıdaki gibi çalıştırılır.

```
$ docker build -f ./builder/Dockerfile ./
```

Burada **-f** parametresi dosyadan oku anlamına gelir. **./** ise çalışma dizinini belirtir. Eğer **-f** verilmemişse çalışma dizininde dockerfile dosyası aranır.

Ayrıca doğrudan git üzerinden de çalıştırılabilir.

```
$ docker build git://gitserver.com/username/repository.git
```

Veya bir tarball indirilerek istenen dockerfile ile çalıştırılması sağlanabilir.

```
$ docker build -f builder/Dockerfile https://example.org/source.tar.gz
```

stdin okunarak çalıştırılabilir.

```
$ cat Dockerfile | docker build -
```

Dockerfile yapısı

Dockerfile dosyaları komutlar yardımı ile çalışır. Aşağıda komut ve kullanım şekli belirtilmiştir.

```
FROM <imaj| scratch>    : hedef imajı kullan veya boş imajla başla
COPY <src> <trgt>       : Çalışma dizinindeki dosyayı kopyalar.
ADD <src> <trgt>        : COPY ile benzerdir fakat arşivleri açarak kopyalar.
RUN <command>           : Komut çalıştırır.
USER <name>             : varsayılan kullanıcı adı belirler
WORKDIR <dir>           : Container içindeki çalışma dizinini belirler.
CMD <command>           : Varsayılan çalıştırılacak olan komutu belirler.
ENV <name> <value>      : Çevresel değişken belirler.
LABEL <key=value>       : Metadata tanımlamak için kullanılır.
EXPOSE <port/protocol>  : Port açmak için kullanılır. protocol kısmı tcp veya udp olabilir.
ARG <name=value>        : ENV ile benzerdir fakat sadece imaj oluşturulurken kullanılabilir.
```

Örneğin aşağıda bir dockerfile dosyası ile kaynak kod derleyelim.

```
FROM alpine
RUN apk add --no-cache build-base
ADD bash-5.0.tar.gz /build
WORKDIR /build/bash-5.0
RUN ./configure --prefix=/usr
RUN make
RUN make install
```

Şimdi bu dosyayı derleme yapmak için kullanalım. Burada **-t** yeni oluşacak imaja isim tag eklemek için kullanılır.

```
$ wget https://ftp.gnu.org/gnu/bash/bash-5.0.tar.gz
$ docker build -t build-bash:5.0 .
```

Git kullanımı

Bu dokümanda sürüm kontrol sistemi olan git komutu nasıl kullanılır anlatılacakdır.

Git kurulumu

git kurulumu debian tabanlı dağıtımlar için aşağıdaki komut kullanılarak kurulabilir.

```
$ apt install git-core
```

Kaynak koddan derlemek için öncelikle git kaynak kodunu indirip bizine açalım. Ardından aşağıdaki adımları uygulayarak derleyelim ve kuralım.

```
./configure --prefix=/usr
make
make install
```

Git ne işe yarar

Git yazdığımız kodların sürüm takibini yapmamızı sağlayan bir araçtır. Bu sayede kod yazarken önceki değişiklikleri kaybetmeden düzenli bir şekilde kodda yaptığımız değişiklikleri görebilir ve ihtiyaç duyulduğunda eski sürümlere dönülebilir. Ayrıca git sayesinde yazdığımız kodu git sunucuları (github, gitlab vb) kullanarak paylaşmak mümkündür.

Git kullanarak kaynak kodun indirilmesi

Git kullanarak kaynak kodu bilgisayarımıza indirmek için *git clone* komutundan faydalanılır. Bu komut git sunucusundan yapılan tüm değişiklikler ile beraber indirir. Eğer belli miktarda değişiklikle indirmek isterseniz **--depth=** parametresi eklenmelidir.

```
# sadece son değişikliği almak için --depth=1 eklendi.
$ git clone https://gitlab.com/sulincix/sayfalar.git --depth=1
```

İndrilen kodun istediğiniz bir dizine indirilmesini istiyorsanız komutun sonuna istenilen konumu yazmanız gereklidir.

```
$ git clone https://gitlab.com/sulincix/sayfalar.git ~/Belgeler/sayfalar
```

Eski değişiklikleri görmek ve eski sürüme dönmek

git log komutunu kullanarak eski değişiklikleri görebilirsiniz. Bu yapılan değişiklikler **commit** olarak adlandırılır. Yazının bundan sonraki kısmında **commit** sözcüğü kullanılacaktır.

Her commitin bir id değeri bulunur. Bu değer kullanılarak eski sürüme dönülebilir.

```
$ git log
-> commit a983f37db618a06a53adb593dd97aa0282775ef5 (HEAD -> master, origin/master, origin/HEAD)
-> Author: aliriza <aliriza.keskin@pardus.org.tr>
-> Date:   Mon Oct 10 08:31:30 2022 +0000
->
->     commit 2
->
-> commit 180a8bcaf81485958fded6a69c97d15161fd1b75
-> Author: aliriza <aliriza.keskin@pardus.org.tr>
-> Date:   Tue Sep 27 10:11:12 2022 +0000
->
->     commit 1
...
```

Burada eski sürüme dönmek için *git reset <commit-id>* komutu kullanılır. Bu komut kodda yapılan değişiklikleri silmeyip git içerisinde eski sürüme dönmeyi sağlar. Eğer git üzerinde yaptığınız değişiklikleri de geri almak isterseniz *--hard* parametresini kullanabilirsiniz. Bu parametre tehlikelidir çünkü yazdığınız kodu geri dönülemez şekilde siler.

Ayrıca commit id yerine *HEAD~n* kullanarak **n** sayıda önceki commite geri dönebilirsiniz.

```
# belirtilen commit-id değerine göre eski sürüme dönmek için (hard reset)
$ git reset 180a8bcaf81485958fded6a69c97d15161fd1b75 --hard
# belirtilen sayıda eski sürüme dönmek için (reset)
$ git reset HEAD~2
```

Sunucudaki güncel değişiklikleri almak

Sunucudaki değişiklikleri *git pull* komutu ile alabiliriz. Bu komut sunucu tarafında yapılan değişiklikleri yereldeki git deposuna ekleyecektir.

```
$ git pull
```

Yeni commit oluşturma

Kaynak kodda yaptığımız değişiklikleri yeni bir commit olarak oluşturmak için *git commit* komutu kullanılır. Bunun için öncelikle hangi dosyaları değiştirdiysek *git add* komutu ile belirtemiz gerekir. Daha sonra *git checkout* komutu ile yapılan değişikliklerin düzgün bir şekilde algılandığından emin olunur. Son olarak *git commit* komutu ile yeni commit oluşturulur.

git commit komutu doğrudan çalıştırıldığında metin düzenleyici ile commit mesajı düzenleme ekranı çalıştırılır. Eğer bu ekranı kullanmak yerine parametre ile belirtmek istereniz **-m** parametresi eklemelisiniz.

```
$ git add rst/git-kullanimi.rst
$ git checkout
-> M    rst/git-kullanimi.rst
$ git commit -m "commit mesajı"
```

commit mesajı düzenleyici **EDITOR** çevreler değişkeni ile belirlenir. Genellikle varsayılan olarak vim kullanılır. Bunu değiştirmek için **~/.bashrc** içerisinde aşağıdaki gibi tanımlama yapabilirsiniz.

```
export EDITOR=nano
```

Commit mesajını değiştirmek için *git commit --amend* komutunu kullanabilirsiniz.

Yeni commit oluşturduktan sonra **HEAD** ve **origin** artık aynı committe olmayacaktır. Burada HEAD sizin yerel olarak bulundurduğunuz halini origin ise git sunucusundaki halini gösterir.

```
$ git log
-> commit 03d5176f5e5b46e43dd688fd7b884a58e60afcd4 (HEAD -> master)
-> Author: aliriza <aliriza.keskin@pardus.org.tr>
-> Date:   Mon Jan 9 11:09:02 2023 +0300
->
->     commit 2
->
-> commit 913d993457d7b07e81746088fbc7cf6aaf9bc01a (origin/master, origin/HEAD)
-> Author: aliriza <aliriza.keskin@pardus.org.tr>
-> Date:   Tue Dec 27 16:44:49 2022 +0300
->
->     commit 1
```

Git sunucusuna gönderme

Yaptığımız değişiklikleri git sunucusuna göndermek için *git push* komutu kullanılır. Sunucu sizden kullanıcı adı ve parola ile doğrulama isteyebilir. Sunucuya ssh anahtarı eklediyseniz ve ssh üzerinden kullanıyorsanız genellikle doğrulama yapılırken parolaya gerek duyulmaz.

Not: github 13 Ağustos 2021 tarihinde https üzerinden commit göndermeyi engellemeye başladı. Parolanız yerine githubdan sağlayacağınız token değerini girmeniz gerekmektedir. Github kullanıyorsanız ssh anahtarı ile kullanmanızı öneririm.

```
$ git push
-> Username for 'https://gitlab.com': sulincix
-> Password for 'https://sulincix@gitlab.com':
```

```
-> Enumerating objects: 3, done.
-> Counting objects: 100% (3/3), done.
-> Writing objects: 100% (3/3), 205 bytes | 205.00 KiB/s, done.
-> Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
-> remote: . Processing 1 references
-> remote: Processed 1 references in total
-> To https://gitlab.com/sulincix/git-dersi.git
->    1ac2e12..2742a1f  master -> master
```

Eğer sunucusunda daha önceden yaptığınız değişiklikler varsa ve sizin yaptığınız değişiklikler ile çakışıyorsa *git push* komutu hata verecektir. Bu duruma **conflict** adı verilir. Conflict çözmek için öncelikle **git pull --rebase** komutu kullanılır.

```
git push
-> Username for ...
-> Password for ...
-> To https://gitlab.com/sulincix/git-dersi.git
->  ! [rejected]        master -> master (fetch first)
```

Yukarıdaki örnekde *git push* komutunu sunucudaki değişiklikleri almadan çalıştırdığımız için bize önce *git pull* komutu kullanarak değişiklikleri almamız söyleniyor.

```
$ git pull --rebase
...
-> From https://gitlab.com/sulincix/git-dersi.git
->    61e3643..e2fe24f  master     -> origin/master
-> Auto-merging commit 3
-> CONFLICT (add/add): Merge conflict in commit 3
-> error: could not apply abaf641... commit 3
...
```

Conflict durumunda **rebase** moduna geçilir. Bu modda çakışan dosyalarda hangisinin seçileceğine karar verilir. Çakışan dosyalar aşağıdaki gibi hal alır. Burada çakışma giderildikten sonra yeni bir commit oluşturmanız gerekmektedir.

```
...
<<<<<<< HEAD
print("hello world")
=======
print("hi world")
>>>>>>> abaf641 (aaa)
...
```

Burada iki değişiklikten hangisinin kalması isteniyorsa o tutulur diğerleri silinir. Daha sonrasında yeni commit oluşturulur Yukarıdaki örnekte son hali aşağıdaki gibi olmalıdır.

```
...
print("hello world")
...
```

Çakışma giderildikten sonra yeni commit oluşturup gönderebiliriz.

```
$ git add main.py
$ git commit -m "Çakışma giderildi"
```

Çakışma giderildikten sonra rebase durumundan çıkmak için *git rebase --continue* komutu kullanılır.

```
$ git rebase --continue
-> Successfully rebased and updated refs/heads/master.
```

Ardından git push komutu ile sunucuya gönderilir.

```
$ git push
-> To https://gitlab.com/sulincix/git-dersi.git
->    e2fe24f..19361f6  master -> master
```

Eğer rebase yapmaktan vazgeçmek istiyorsanız *git rebase --abort* kullanmanız gerekir. Bu sayede rebase işleminden çıkılır.

Eğer sunucuya değişiklikleri zorla göndermek için **--force** parametresi kullanılır. Bu işlem sunucudaki değişiklikleri silip yerine yereldeki değişikliklerin atılmasını sağlar.

Not: Bu işlem sonucunda sunucuda bulunan değişiklikler silindiği için tehlikelidir. **Daha önemlisi arkadaşlarınız size küfür edebilir :D** Mümkünse hiç force push yapmayın.

Branch kavramı

Git üzerinde birden çok dal ile çalışmak mümkündür. Bu dallar **branch** sözcüğü ile ifade edilir. Bu sadece koda yeri bir özelliği geliştirirken farklı bir dal kullanıp kodun stabil çalışan halini kullancak kişiler için korumanız mümkündür.

Mecut branchları görüntülemek için *git branch* komutu kullanılır. Varsayılan branch adımız genellikle **master** olarak tanımlıdır.

```
$ git branch
-> * master
```

Yeni bir branch oluşturmak için *git branch <dal-adı>* komutu kullanılır.

```
# yeni branch oluşturalım
$ git branch development
# branch listeleyelim
$ git branch
->   development
-> * master
```

Yukarıdaki örnekte mevcut bulunduğumuz branch başında * işareti bulunmaktadır. Bulunduğumuz branchı değiştirmek için *git switch <dal-adı>* komutu kullanılır.

```
$ git switch development
-> Switched to branch 'development'
$ git branch
-> * development
->   master
```

Dallarda yapılan değişiklikleri birleştirmek için *git merge <dal1> <dal2>* komutu kullanılır.

```
$ git merge development master
```

Sunucuya değişikliklerimizi istenilen dalda göndermek için *git push <remote-adı> <dal-adı>* kullanılır.

```
# master branchına geçelim
$ git switch master
# development branchını sunucuya yollayalım.
$ git push origin development
```

Branch silmek için *git branch -d <dal-adı>* komutunu kullanabilirsiniz. Bulunduğunuz dalı silemezsiniz. (Bindiğiniz dalı kesemediğiniz gibi:D)

```
# önce diğer brancha geçelim
$ git switch master
# development branchını silelim
$ git branch -d development
```

Bir branchı yeniden adlandırmak için *git branch --move <eski-ad> <yeni-ad>* komutu kullanılır.

```
# bir branch oluşturalım.
$ git branch dev
# yeniden adlandıralım.
$ git branch --move dev development
```

Remote kavramı

Git üzerinde birden çok sunucu tanımlanabilir ve bunlardan istenilene veri alınıp verilebilir. Bu sunucular **remote** sözcüğü ile ifade edilir.

Mevcut remote listesi için *git remote* komutu kullanılır. varsayılan remote adı genellikle **origin** olarak tanımlanır.

```
$ git remote
-> origin
```

Bir remotenin hangi adreste olduğunu öğrenmek için *git remote get-url <remote-adı>* komutu kullanılır.

```
$ git remote get-url origin
-> https://gitlab.com/sulincix/sayfalar.git
```

Remotenin adresini değiştirmek için ise *git remote set-url <remote-adı> <yeni-adres>* kullanılır.

```
$ git remote set-url origin https://gitlab.com/sulincix/git-dersi.git
```

Yeni bir remote eklemek için ise *git remote add <remote-adı> <adres>* kullanılır.

```
$ git remote add github https://github.com/sulincix/sayfalar.git
$ git remote
```

```
-> origin
-> github
```

Remote üzerinden değişiklikleri alıp vermek için *git pull <remote-adı>* ve *git push <remote-adı>* kullanılır.

```
# Değişiklikleri alalım
$ git pull github
# diğer remote üzerine gönderelim.
$ git push origin
```

Bir remoteyi silmek için *git remote remove <remote-adı>* komutu kullanılır. Yeniden adlandırmak için ise *git remote rename <eski-ad> <yeni-ad>* komutu kullanılır.

```
# yeniden adlandıralım.
$ git remote rename github git
# remote silelim
$ git remote remove git
```

Squash commit kavramı

Bazen git üzerinde farklı bir branch üzerinde geliştirme yaparken çok fazla miktarda commit ürettiğinizde bunları ana branch üzerine birleştirirken birsürü committen oluşması yerine tek bir commit haline getirmek isteyebilirsiniz. Bu gibi durumlarda commitleri birleştirerek **squash commit** elde edebilirsiniz. Bunun için rebase module geçmemiz gerekmektedir.

İlk olarak *git rebase -i <commit-id>* komutu ile rebase moduna geçelim. burada **-i** parametresi commitleri birleştirmemiz için metin düzenleyicimizde bir ekran açacaktır.

```
# rebase moduna geçelim.
$ git rebase -i HEAD~5
# metin düzenleyicimizde aşağıdaki gibi metin bulunur.
pick aa34d35 commit 5
pick 879917e commit 4
pick 864dc97 commit 3
...
```

Yukarıdaki örnekte **pick** ile belirtilen commitleri **squash** olarak değiştirirseniz commit bir önceki commit ile birleştirilmiş olur. Diğer komutlar düzenleyicide altta açıklama satırı olarak yer almaktadır.

Düzenleyicide kaydedip çıktıktan sonra bu sefer commit mesajı ekranı ile karşılaşırız. Burada birleştirilmiş commit mesajını yazıp kaydettikten sonra commitler birleştirilmiş olur.

makefile dersi

makefile formatı yazılan bir kaynak kodu derlemek ve yüklemek için kullanılan ne yaygın derleme talimatı formatlarından biridir.

Bu yazıda sizlere makefile dosyası nasıl yazılır anlatacağım.

Genel bakış

Örneğin aşağıdaki gibi bir **C** kodumuz olsun

```
#include <stdio.h>
int main(){
    puts("Hello world!\n");
    return 0;
}
```

Bunu aşağıdaki komutu kulanarak derleriz ve kurarız.

```
$ gcc -o hello hello.c
$ install hello /usr/bin/hello
```

Makefile dosyalarının bölüm tanımlamalarında girintileme amaçlı **Tab** kullanılır.

Şimdi aşağıdaki makefile dosyasını inceleyelim.

```
PREFIX=/usr
build:
        $(CC) -o hello hello.c

install:
        install hello $(DESTDIR)/$(PREFIX)/bin/hello
```

Burada **PREFIX**, **CC**, **DESTDIR** gibi parametreler değişkendir. Bu değişkenler derleme esnasında değiştirilebilir.

Bu makefile dosyasını kullanarak derlemeyi ve yüklemeyi aşağıdaki gibi yaparız.

```
$ make
$ make install
```

Görüldüğü gibi derleme ve yükleme işlemi daha kolay ve nasıl derleneceğini basitçe belirtmiş olduk.

Burada kullandığımız değişkenler şu şekilde açılnabilir.

- PREFIX = /usr olarak tanımladık.
- DESTDIR = paket sistemleri paket yaparken bu değişkeni otomatik olarak değiştirir. Kurulacak kök dizin konumudur.
- CC = derleyicinin adıdır. Bu değişkeni ayarlayarak derleyiciyi değiştirebilirsiniz.

Make komutuna eğer hiç parametre verimezsek ilk baştaki bölümü çalıştırır. Biz ilk başta **build** tanımladığımız için make komutu build çalıştırır. make komutuna parametre olarak bölüm verirsek o bölüm çalıştırılır.

Değişken işlemleri

Değişken tanımlamak için **variable=value** şeklinde tanımlanabilir. değişkeni kullanırken de **$()** işareti arasına alınır. Örneğin:

```
yazi=hello world
hello:
        echo $(yazi)
```

Bu değişkeni **make yazi=deneme123** şeklinde komut vererek değiştirebiliriz.

Var olan bir değişkene ekleme yapmak için **+=** ifadesi kullanılır. **:=** ifadesi eğer tanımlama varsa ekleme yapar. **?=** sadece daha önceden tanımlanmışsa ekleme yapar.

```
yazi=hello
yazi+=world
sayi:=$(shell ls)
hello:
	echo $(yazi)
```

Eğer **$** işareti kullanmanız gereken bir durum oluşursa **$$** ifadesi kullanabilirsiniz. Örneğin:

```
hello:
	bash -c "echo $$HOME"
```

Bölümler

Makefile yazarken bölümler tanımlanır ve eğer bölümün adı belirtilmemişse ilk bölüm çalıştırılır. Bölümler arası bağımlılık vermek için aşağıdaki gibi bir kullanım yapılmalıdır.

```
yazi: sayi test
	echo "Hello world"
sayi:
	echo 12
test:
	echo test123
```

Yukarıdaki dosyayı çalıştırdığımızda sırasıyla **sayi** -> **test** -> **yazi** bölümleri çalıştırılır.

Aynı işi yapan birden çok bölüm şu şekilde tanımlanabilir.

```
bol1 bol2:
	echo Merhaba
# Şuna eşittir.
bol1:
	echo Merhaba
bol2:
	echo Merhaba
```

Bölümün adını **$@** kullanarak öğrenebiliriz.

```
bolum:
	echo $@
```

Bölümün tüm bağımlılıklarını almak için için **$^** kullanabiliriz.

```
bolum: bol1 bol2
	echo $^
bol1 bol2:
	true
```

$?** ifadesi **$^ ile benzerdir fakat sadece geçerli bölümden sonra tanımlanan bölümleri döndürür.

```
bol1:
        true
bolum: bol1 bol2
        echo $?
bol2:
        true
```

$< ifadesi sadece ilk bağımlılığı almak için kullanılır.

```
bol1 bol2:
        true
bolum: bol1 bol2
        echo $<
```

Eğer **xxxx.o** şeklinde bir kural tanımlarsanız bu kural çalıştırıldıktan sonra gcc ile kural adındaki dosya derlenir.

```
main: main.o
main.o: main.c test.c

main.c:
        echo "int main(){}" > main.c
%.c:
        touch $@
```

Burada main.c dosyası var olmayan bir dosyadır ve derleme esnasında oluşturulur. test.c dosyası ise daha önceden var olan bir dosyadır ve o dosyaya bir şey yapılmaz. main.c kuralı sadece main.c için çalıştırılırken **%.c** şeklinde belirtilen kular hem main.c hem test.c için çalıştırılır. **main** ile belirttiğimiz kuralda main.o bağımlılığı olduğu için bi derlemenin sonucu olarak main adında bir derlenmiş dosya üretilmektedir.

wildcard ve shell

Wildcard ifadesi eşleşen dosyaları döndürür.

```
files := $(wildcard *.c)
main:
        gcc -o main $(files)
```

Shell ifadesi ise komut çalıştırarak sonucunu döndürür.

```
files := $(shell find -type f -iname "*.c")
main:
        gcc -o main $(files)
```

Birden çok dosya ile çalışma

make -C xxx şeklinde alt dizindeki bir makefile dosyasını çalıştırabilirsiniz.

```
build:
        make -C src
```

Ayrıca **include** kullanarak başka bir dosyada bulunan kuralları kullanabilirsiniz.

```
# Makefile dosyası
include build.mk
build: main
        gcc main.c -o main
# build.mk dosyası
main:
        echo "int main(){return 0;}" > main.c
```

Koşullar

ifeq ifadesi ile koşul tanımlanabilir. aşağıdaki ifadeşi **make CC=clang** şeklinde çalıştırırkanız clang yazdırır, parametresiz bir şekilde çalıştırırsanız gcc yazdırır. Burada dikkat edilmesi gereken konu **ifeq**, **else**, **endif** girintilenmeden yazılır.

```
build:
ifeq ($(CC),clang)
        echo "clang"
else
        echo "gcc"
endif
```

Komut özellik ifadeleri

Eğer komutun başına **@** işareti koyarsanız komut ekrana yazılmadan çalıştırılır. **-** yazarsanız komut hata alsa bile geri kalan kısımlar çalışmaya devam eder.

```
build:
        @echo "Merhaba dünya"
        -gcc main.c -o main
```

while ve for kullanımı

Bash betiklerinde kullandığımız for ve while yapısı makefile yazarken aşağıdaki gibi kullanılabilir. done dışındaki satırların sonuna **** işareti eklenir, do dışındaki satırların sonuna da **;** işareti koyulur.

```
build:
        @for sayi in 1 2 3 $(dizi) ; do \
            echo $$sayi ; \
            echo "diger satir" ; \
        done
```

SHELL değişkeni

SHELL değişkeni makefile altındaki komutların hangi shell kullanılarak çalıştırılacağını belirtir. Varsayılan değeri **/bin/sh** olarak belirlenmiştir. Örneğin debian tabanlı dağıtımlarda /bin/sh konumu /bin/dash bağlıyken archlinuxta /bin/bash bağlıdır. **dash [[** kullanımını desteklemezken **bash** destekler. Bu sebeple uyumluluğu arttırmak için **SHELL** değişkenini zorla /bin/bash olarak değiştirebiliriz. Aşağıdaki örnekle konuyu daha iyi anlamak için SHELL değişkenini python3 ayarladık ve python kodu yazdık.

```
SHELL=/usr/bin/python3
build:
```

```
        import os ;\
        liste = os.listdir("/") ;\
        print(liste[0])
```

Python dersi

Bu yazıda python programlama dilini hızlıca anlatacağım. Bu yazıda karıştırılmaması için girdilerin olduğu satırlar <- ile çıktıların olduğu satırlar -> ile işaretlenmiştir.

Açıklama satırları

Python programlama dilinde açıklamalar **#** işaretinden sonrası için geçerlidir. Örneğin:

```
#bu bir açıklama satırıdır.
```

Bununla birlikte çoklu açıklama satırı yapmak için **"""** işareti arasına alabiliriz.

```
""" Bu bir açıklama satırı
Bu diğer açıklama satırı
Bu son açıklama satırı """
```

Temel bilgiler

Python programlarının ilk satırında **#!/usr/bin/python3** satırı bulunmalıdır. Bir python programını çalıştırmak için Şunları uygulamamız gereklidir.

```
#!/usr/bin/python3
# Çalıştırma izni vererek çalıştırabiliriz.
chmod +x dosya.py
./dosya.py
# Veya doğrudan çalıştırabiliriz.
python3 dosya.py
```

Python programlama dilinde satır sonuna **;** koyulmaz.

Python programlarında işler işlevler üzerinden yürür. işlevlerin girdileri ve çıktıları bulunur.

```
cikti = islev(girdi1, girdi2)
```

Pythonda girintileme olayı için de **{** ve **}** kullanılmak yerine boşluklandırma kullanılır. Herhangi bir girintilemeye başlanan satırın sonunda **:** işareti bulunur. Örneğin:

```
f = 12 # f sayısını 12ye eşitledik
if f == 12: # f sayısı 12ye eşit mi diye sorguladık
    print("eşit") # ekrana yazı yazdırık
```

Girintileme için 4 boşluk, 2 boşluk veya tek tab kullanabilirsiniz. Bu yazıda 4 boşluğu tercih edeceğiz.

Yazı yazdırma

Pythonda ekrana yazı yazmak için **print** işlevini kullanıyoruz.

```
print("Merhaba Dünya")
-> Merhaba Dünya
```

Birden çok ifadeyi yazdırmak için **print** işlevine birden çok girdi verebilirsiniz. Bu şekilde aralarına birer boşluk koyarak yazdırır.

```
# Yazılar tırnak içine alınır.
# Sayılar tırnak içine alınmaz.
# True ve False doğruluk belirtir.
print("Merhaba",12,"Dünya",True)
-> Merhaba 12 Dünya True
```

Değişkenler

Değişkenler içerisinde veri bulunduran ve ihtiyaç durumunda bu veriyi düzenleme imkanı tanıyan kavramlardır. Değişkenler tanımlanırken **degisken = deger** şeklinde bir ifade kulanılır.

```
i = 12
yazi = "merhaba dünya"
k = 1.2
hmm = True
```

Değişken adları sayı ile başlayamaz, Türkçe karakter içeremez ve sadece harfler, sayılar ve **-** **_** karakterlerinden oluşur.

Değişkenler kullanılırken başına herhangi bir işaret almasına gerek yoktur. Örneğin:

```
i = 12
print(i)
-> 12
```

Değişkenler tanımlanırken her ne kadar türlerini belirtmesek bile birer türe sahip olarak tanımlanır. Bunlar başlıca **integer**, **float**, **string**, **boolean** türleridir.

Bir değişkenin türünü öğrenmek için **type** işlevini kullanabiliriz.

```
veri = "abc123"
turu = type(veri)
print(turu)
-> <class 'str'>
```

Boş bir değişken tanımlamak için onun değerine **None** atayabiliriz. Bu sayede değişken tanımlanmış fakar değeri atanmamış olur.

```
veri = None
tur = type(veri)
print(tur)
-> <class 'NoneType'>
```

String

String türünden değişkenler yazı içerir. " veya ' veya """ arasına yazılırarak tanımlanır.

```
yazi1 = "merhaba"
yazi2 = 'yazım'
yazi3 = """dünya"""
```

String türünden değişkenler **+** işareti ile uc uca eklenebilir.

```
yazi = "merhaba" + ' ' + """dünya"""
print(yazi)
-> merhaba dünya
```

Değişkeni birden çok kez toplamak için * işareti kullanılabilir.

```
yazi = "ali"*5
print(yazi)
-> alialialialiali
```

String türünden bir değişkenin içerisindeki bir bölümü başka bir şey ile değiştirmek için **replace** işlevi kullanılınabilir.

```
veri = "Merhaba"
veri2 = veri.replace("rha","123")
print(veri2)
-> Me123ba
```

Integer

Integer türünden değişkenler tam sayı belirtir. Dört işlem işaretleri ile işleme sokulabilirler.

```
sayi = (((24/2)+4)*2)-1
"""
24/2 = 12
12+4 = 16
16*2 = 32
32-1 = 31
"""
print(sayi)
-> 31
```

Integer değişkenlerin kuvvetlerini almak için ** kullanılır.

```
sayi = 2**3
print(sayi)
-> 8
```

String türünden bir değişkeni integer haline getirmek için **int** işlevi kullanılır.

```
print(int("12")/2)
-> 6
```

Float

Float türünden değişkenler virgüllü sayılardır. Aynı integer sayılar gibi dört işleme sokulabilirler. İki integer değişkenin birbirine bölümü ile float oluşabilir.

```
sayi = 1/2 # sayi = 0.5 şeklinde de tanımlanabilir.
print(sayi)
-> 0.5
```

Bir float değişkenini integer haline getirmek için **int** işlevi kullanılır. Bu dönüşümde virgülden sonraki kısım atılır.

```
sayi = 3.2
print(sayi)
sayi2 = int(3.2)
print(sayi2)
-> 3.2
-> 3
```

Not: float ile string çarpılamaz.

String türünden bir değişkeni float haline getirmek için **float** işlevi kullanılır.

```
print(float("2.2")/2)
-> 1.1
```

Boolean

Boolean değişkenler sadece **True** veya **False** değerlerini alabilir. Bu değişken daha çok koşullarda ve döngülerde kullanılır. iki değişkenin eşitliği sorgulanarak boolean üretilebilir.

```
bool = 12 == 13
"""
== eşit
!= eşit değil
<  küçük
>  büyük
<= küçük eşit
>= büyük eşit
"""
print(bool)
-> False
```

boolean değişkeninin tersini almak için **not** ifadesi kullanılabilir.

```
veri = not True
print(veri)
-> False
```

Bir string türünden değişkenin içinde başka bir string türünden değişken var mı diye kontrol etmek için **in** ifadesi kullanılır. Bu ifadenin sonucu boolean üretir.

```
veri = "ef" in "Dünya"
veri2 = "ny" in "Dünya"
print(veri,veri2)
-> False True
```

Boolean değişkenlerde mantıksal işlemler **and** ve **or** ifadeleri ile yapılır.

```
veri = 12 < 6 or 4 > 2 # False or True = True
print(veri)
-> True
```

Klavyeden değer alma

Python programlarının kullanıcı ile etkileşime girmesi için klavye üzerinden kullanıcıdan değer alması gerekebilir. Bunun için **input** işlevi kullanılır. Bu işlevin çıkışı string türündendir.

```
a = input("Bir değer girin >")
print(a,type(a))
<- 12
-> 12 <class 'str'>
```

String türünden bir ifadeyi bir değişken üretmek için kullanmak istiyorsak **eval** işlevini kullanabiliriz.

```
a = eval("12/2 == 16-10") # string ifade çalıştırılır ve sonucu aktarılır.
print(a)
-> True
```

Not: Bu işlev tehlikelidir. Potansiyel güvenlik açığına neden olabilir! Mümkün olduğu kadar kullanmayın :D

Koşullar

Koşul tanımı yapmak için **if** ifadesi kullanılır. Koşul sağlanmıyorsa **elif** ifadesi ile yeni bir koşul tanımlanabilir veya **else** ifadesi ile koşulun sağlanmadığı durum tanımlanabilir.

```
if koşul:
    eylem
elif koşul:
    eylem
else:
    eylem
```

Örneğin bir integer değişkenin çift olup olmadığını bulalım.

```
if 13 % 2 == 0 : # % işareti bölümden kalanı bulmaya yarar.
    print("Çift sayı")
else:
    print("Tek sayı")
```

Değeri olmayan (None) değişkenler koşul ifadelerinde **False** olarak kabul edilir.

```
veri = None
if veri:
    print("Tanımlı")
else:
    print("Tanımsız")
-> Tanımsız
```

Koşul tanımlamayı alternatif olarak şu şekilde de yapabiliriz:

```
koşul and eylem
""" Şununla aynıdır:
if koşul:
    eylem
"""
koşul or eylem
""" Şununla aynıdır:
if not koşul:
    eylem
"""
```

Bu konunun daha iyi anlaşılması için:

```
12 == 12 and print("eşittir")
12 == 14 or print("eşit değildir")
-> eşittir
-> eşit değildir
```

Diziler

Diziler birden çok elemanı içerebilen değişkenlerdir. Diziler aşağıdaki gibi tanımlanır:

```
a = [1, 3, "merhaba", True, 1.2, None]
```

Dizilerin elemanlarının türü aynı olmak zorunda değildir. Hatta None bile olabilir.

Dizilerde eleman eklemek için **append** veya **insert** işlevini eleman silmek için ise **remove** veya **pop** işlevi kullanılır. Örneğin:

```
a = [22]
print(a)
a.append("Merhaba") # Sona ekleme yapar.
a.insert(0,12) # 0 elemanın ekleneceği yeri ifade eder.
print(a)
a.remove(22) # 22 elemanını siler
print(a)
a.pop(0) # 0. elemanı siler.
print(a)
-> [22]
-> [12, 22, 'Merhaba']
-> [12, 'Merhaba']
-> ['Merhaba']
```

Dizileri sıralamak için **sort** boşaltmak için ise **clear** işlevi kullanılır. Bir dizinin istenilen elemanını öğrenmek için **liste[index]** şeklinde bir ifade kullanılır. Index numaraları 0dan başyan integer olmalıdır. negatif değerlerde sondan saymaya başlar.

```
a = [1, 3, 6, 4, 7, 9, 2]
print(a[2],a[-3])
a.sort()
print(a)
a.clear()
print(a)
-> 3 7
```

```
-> [1, 2, 3, 4, 6, 7, 9]
-> []
```

Dizideki bir elemanın uzunluğunu bulmak için **len** işlevi, elemanın dizinin kaçıncısı olduğunu bulmak için ise **index** işlevi kullanılır.

```
a = [12, "hmm", 3.2]
sayi = len(a)
sayi2 = a.index(3.2)
print(sayi,sayi2)
-> 3
-> 2
```

Dizilerin elemanlarını **+** kullanarak birleştirebiliriz.

```
a = [1, 2, 3]
b = [4, 5, 6]
c = a + b
print(c)
-> [1, 2, 3, 4, 5, 6]
```

Dizilerin bir bölümünü aşağıdakine benzer yolla kesebiliriz:

```
a = [1, 3, 5, 7, 9, 12, 44, 31, 16]
b = a[:2] # baştan 3. elemana kadar.
c = a[4:] # 4. elemandan sonrası
d = a[3:6] # 4. elemandan 6. elemana kadar (dahil)
```

String türünden bir değişkeni belli bir harf veya harf öbeğine göre bölmek için **split** işlevini kullanırız. Ayrıca string türünden bir değişkenin başındaki ve sonundaki boşlukları temizlemek için **strip** işlevini kullanırız.

```
veri="   Bu bir yazıdır   "
veri2 = veri.strip()
print(len(veri),len(veri2))
liste = veri2.split(" ")
print(liste)
-> 20 14
-> ['Bu', 'bir', 'yazıdır']
```

While döngüsü

Döngüler belli bir işi koşul bağlanana kadar tekrar etmeye yarayan işlevdir. Kısaca **while** döngüsü ile **if** arasındaki fark **while** içerisindeki durum tamamlandığı zaman tekrar başa dönüp koşulu kontrol eder.

```
while koşul:
    eylem
```

Örneğin 1den 10a kadar olan sayıları yazalım. Bu durumda *i* sayısı 10 olana kadar sürekli olarak ekrana yazılıp değeri 1 arttırılacakdır.

```
i = 1
while i < 10:
    print(i)
    i+=1 # i = i + 1 ile aynı anlama gelir.
-> 1 2 3 4 5 6 7 8 9 (Bunu alt alta yazdığını hayal edin :D )
```

Bir döngüden çıkmak için **break** ifadesi kullanılır. Döngünün o andi adımını tamamlayıp diğer adıma geçmek için ise **continue** ifadesi kullanılır.

Örneğin aşyağıda siz çift sayı girene kadar sürekli olarak çalışan bir program yazalım.

```
while True:
    sayi = int(input("Sayı girin"))
    if sayi % 2 == 0:
        break
```

For döngüsü

For döngüsü while ile benzerdir fakat koşul aranmak yerine iteration yapar. Bu işlemde bir dizinin bütün elemanları tek tek işleme koyulur. Aşağıdaki gibi bir kullanımı vardır:

```
for eleman in dizi:
    eylem
# Şununla aynıdır
i = 0
toplam = len(dizi)
while i < toplam: # eleman yerine dizi[i] kullanabilirsiniz.
    eylem
    i += 1
```

Örneğin bir integer değişkenlerden oluşan dizi oluşturalım ve elemanlarını 2ye bölerek ayrı bir diziye ekleyelim.

```
a = [2, 4, 6, 8, 10] # dizi tanımladık
b = [] # diğer diziyi tanımladık
for i in a: # a elemanları i içine atılacak
    b.append(i/2) # b içine elemanın yarısını ekledik.
print(b)
-> [1, 2, 3, 4, 5]
```

Eğer dizi yerine string türünden bir değişken verirsek elemanlar bu stringin harfleri olacaktır. Aşağıdaki örnekte string içerisinde kaç tane a veya e harfi bulunduğunu hesaplayalım.

```
veri = "Merhaba Dünya"
toplam = 0
for i in veri:
    if i == "a" or i == "e":
        toplam += 1
print(toplam)
-> 4
```

Şimdiye kadarki anlatılanların daha iyi anlaşılması için asal sayı hesaplayan bir python kodu yazalım:

```
asallar = [2] # ilk asal sayıyı elle yazdık.
i = 3 # Şu anki sayı
while i < 60: # 60a kadar say
    hmm = True # asal sayı mı diye bakılan değişken
    for e in asallar: # asal sayılar listesi elemanları
        if i % e == 0: # tam bölünüyor mu
            hmm = False # asal sayı değildir
            break # for döngüsünden çıkmak için
    if hmm: # Asal sayıysa diziye ekleyelim
        asallar.append(i)
    i += 1 # mevcut sayımızı 1 arttıralım.
print(asallar) # 60a kadar olan asal sayılar dizisini yazalım.
-> [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59]
```

İşlevler

Python programlarken işlev tanımlayıp daha sonra bu işlevi kullanabiliriz. İşlevler aşağıdaki gibi tanımlanırlar:

```
def islev(girdi1,girdi2):
    eylem
```

işlevlerde çıktı sonucu olarak bir değişken döndürmek için **return** ifadesi kullanılır. Örneğin girdideki sayıları toplayan işlev yazalım.

```
def topla(sayi1,sayi2):
    return sayi1 + sayi2
    print("Merhaba") # bu satır çalıştırılmaz
toplam = topla(3,5)
print(toplam)
-> 8
```

Eğer bir değişken sadece işlevin içerisinde tanımlanırsa o değişken işlevin dışında tanımsız olur.

```
def yazdir():
    yazi = "Merhaba"
    print(yazi)
yazdir()
print(yazi)
-> Merhaba
-> Traceback (most recent call last):
->   File "ders.py", line 5, in <module>
->     print(yazi)
-> NameError: name 'yazi' is not defined
```

Bir işlevin ne işe yaradığını öğrenmek için **help** işlevi kullanılır. işlevin ne işe yaradığını tanımlamak için ise ilk satıra """ içerisine yazabiliriz. Bunu tanımlamak programınızı inceleyen diğer insanlar için yararlı olacaktır.

```
def abc(sayi):
    """Girilen sayıyı 10dan çıkartır"""
    return 10-sayi
help(abc)
```

```
->  Help on function abc in module __main__:
->
->  abc(sayi)
->      Girilen sayıyı 10dan çıkartır
```

Bir işlevin birden çok çıktısı olabilir. Bunun için **return** ifadesini virgülle ayrılmış olarak birden çok değişken ile kullanmalıyız.

```
def yer_degistir(a,b):
    """Girilen değişkenlerin yerini değiştirir"""
    return b,a
c = 12
d = 31
c,d = yer_degistir(c,d)
# Bunun yerine doğrudan c,d = d,c kullanılabilirdi.
print(c,d)
-> 31 12
```

Konunun daha iyi anlaşılabilir olması için girilen dizinin sayılarının ortalamasını alan bir fonksiyon yazalım.

```
def ortalama(dizi):
    toplam = 0 # toplam değişkeni tanımladık.
    for eleman in dizi: # for döngüsü oluşturduk.
        toplam += int(eleman) # elemanları topladık.
    return toplam / len(dizi) # toplamı eleman sayısına böldük.

ort = input("Dizi giriniz. aralarına , koyunuz")
print(ortalama(ort.split(",")))
<- 1,34,22,-32
-> 6.25
```

Sınıflar

Sınıf kavramı işlevlerin ve değişkenlerin guruplanarak nesneler haline getirilmesinden meydana gelir. Yani bir sınıf ona bağlı işlevlerden ve değişkenlerden oluşur. Sınıflar aşağıdaki gibi tanımlanırlar.

```
class sinif:
    def __init__(self,girdi):
        eylem
    def islev(self,girdi):
        eylem
```

Burada **__init__** işlevi sınıfı oluştururken çalıştırılan ilk eylemleri tanımlamak için kullanılır. sınıf işlevleri tanımlanırken ilk girdi olarak **self** kullanılmalıdır. Bu ifade sınıfın kendisi anlamına gelir. Örneğin bir sınıf tanımlayalım ve bu sınıftaki işlevler ile girdideki sayılara toplama ve çıkartma işlemi uygulayalım.

```
class sayi_isle:
    def __init__(self,ilk,ikinci):
        self.sayi1 = ilk
        self.sayi2 = ikinci
```

```
    def topla():
        return self.sayi1 + self.sayi2
    def cikart():
        return self.sayi1 - self.sayi2

nesne = sayi_isle(12,3)
a = nesne.topla()
b = nesne.cikart()
print(a,b)
-> 15 9
```

Burada işlevlere **nesne.islev()** ifadesi ile erişebiliyoruz. Aynı zamanda değişkenlere de **nesne.degisken** ifadesi ile erişmemiz ve değiştirmemiz mümkündür. Sınıf içerisinden ise **self.islev()** ve **self.degisken** şeklinde bir ifade kullanmamız gerekmektedir.

Dosya işlemleri

Bir dosyayı açmak için **open** işlevi kullanılır. Açılan dosyadan satır okumak için **readline** işlevi, tamamını okumak için **read** işlevi, tüm satırları okuyup dizi haline getirmek için ise **readlines** işlevi kullanılır.

deneme.txt adında içeriği aşağıdaki gibi olan bir doysamız olsun:

```
Merhaba dünya
Selam dünya
sayı:123
```

Aşağıdaki örnekte bu dosyayı açıp okuyup ekrana basalım.

```
dosya = open("dosya.txt","r") # okumak için r kullanılır.
ilksatir = dosya.readline()
tumu = dosya.read()
satirlar = dosya.readlines()
print(len(satirlar))
-> 3
```

Dosyaya yazmak için ise **write** işlevi kullanılır. Okuma ve yazma işlemleri bittikten sonra **close** işlevi ile dosya kapatılmalıdır. Dosyayı kapatmadan değişiklikleri diske işletmek için **flush** işlevi kullanılır.

```
dosya = open("dosya.txt","w") # yazmak için w eklemek için a kullanılır.
dosya.write("Merhaba dünya\n")
dosya.write("Selam dünya\n")
dosya.write("sayı:123\n")
dosya.close()
```

Modüller

Python programlarında kodların markaşıklaşmasını önlemek ve daha kullanışlı hale getirmek amacıyla modüller bulunur. Modüller **import** ifadesi ile çağırılır. Modüller aslında birer Birer python kütüphanesidir ve sınıf sayılırlar. Örneğin deneme.py dosyamızda aşağıdaki kodlar bulunsun:

```
yazi = "Merhaba"
sayi = 12
def yazdir():
    print(yazi)
class sinif:
    def islev(self):
        print("selam")
```

Şimdi bu modülümüzü çağırıp içerisindeki işlevleri ve değişkenleri kullanalım.

```
import deneme # deneme modülünü çağırdık
print(deneme.yazi) # değişkeni kullandık
deneme.yazdir() # işlevi kullandık.
deneme.sayi = 76 # değişkeni değiştirdik
nesne = deneme.sinif() # sınıftan nesne oluşturduk
nesne.islev() # nesneyi kullandık
-> Merhaba
-> selam
```

Şimdiye kadar anlatılanların daha iyi anlaşılması için aşağıda ini parser örneği yapalım. Örnek bir ini dosyası aşağıdaki gibidir:

```
[bölüm1]
veri1=deger1
veri2=deger2
[bölüm2]
veri3=deger3
veri4=deger4
```

Bir adet modül yazıp bu modül ile ini dosyası okuyup istenilen bölümdeki değeri bulalım ve döndürelim.

```
[Merhaba]
dünya=12
selam=44
[hmm]
veri=abc123x
sayı=44
```

```
# iniparser.py içeriği
dosya = None # boş dosya nesnesi
class inidosya:
    def __init__(self,yol):
        ini = open(yol,"r") # ini dosyasını açtık
        self.icerik = ini.read() # dosya içeriğini okuduk
    def deger_al(bolum,veri):
        etkin = False # istenilen yere gelene kadar etkisiz kal
        for satir in dosya.icerik.split("\n"):
            if "[" + bolum + "]" in satir: # okunan satırda istenilen bölümün başı mı
                etkin = True # etkinleştir
            if etkin and "=" in satir: # etkinse ve satırda = bulunuyorsa
                if satir.split("=")[0] == veri: # = işaretine göre 0. eleman aranan mı
                    return satir.split("=")[0] # = işaretine göre böl . elemanı al
```

```
# main.py dosyası içeriği
import iniparser # modülü yükle
iniparser.dosya = iniparser.inidosya("dosya") # ini dosyasını yükle
deger=iniparser.deger_al("hmm","veri") # değeri al
print(deger.strip()) # değerin başında sonunda boşluk varsa sil ve yaz
-> abc123x
```

Bir modülü diğer bir modülün genişletilmişi olarak tanımlayabiliriz. Gelişletilen modül asıl modüldeki tüm fonksiyonlara ve değişkenlere sahip olur. Aşağıdaki örnekteki gibi **super().__init__()** kullanarak üst modülümüzdeki tüm tanımlamalara sahip olmasını sağlayabiliriz.

```
# ornek.py dosyası
class deneme:
    def __init__(self):
        self.sayi = 13
    def hmm(self,yazi):
        print(yazi)

class genis(deneme):
    def __init__(self):
        super().__init__()
        self.sayi2 = 44
```

```
# main.py dosyası
from ornek import genis as g # ornek.py dosyasındaki genis sınıfını g olarak içeri aldık.
print(g.sayi, g.sayi2)
g.hmm("abc123")
-> 13 44
-> abc123
```

Vala dersi

Bu yazıda vala programlama dilini anlatacağım. Örneklende **...** bulunan satırlar üstünde ve altında başka kodların bulunabileceğini belirtir. Çalıştırılan komutların başında **$** işareti kullanılmıştır.

Vala programlama dili derlemeli bir dil olup yazmış olduğunuz kod valanın çeviricisi yardımı ile önce **C** koduna çevrilir ve ardından **gcc** veya **clang** gibi bir derleyici kullanılarak derlenir. Vala kaynak kodunu derlemek için **valac** kullanılır. Örneğin aşağıda basit bir derleme örneği verilmiştir.

```
$ valac main.vala
```

Vala programlarını main fonksiyonu ile başlar. Bu fonksiyon aşağıdaki gibi yazılır:

```
int main(string[] args){
   ...
   return 0;
}
```

Burada **int** fonksiyonun çıktı türünü **main** adını **string[] args** ise parametresini ifade eder. **return 0** ise çıkış kodunu bildirir. Vala programlama dilinde her satırın sonunda **;** bulunmak zorundadır.

Girintileme

Vala girintilemeye dikkat eden bir dil değildir fakat kodun okunaklı olması açısından girintilemeye dikkat etmenizi öneririm. Girintileme ile ilgili aşağıdaki kuralları takip edebilirsiniz. Bu kurallar zorunluluk oluşturmadığı gibi kendinize göre farklılaştırabilirziniz.

- Koşullar döngüler ve fonksiyonlarda 4 boşluk veya tek tab ile girintileyin.
- Virgülden sonra bir boşluk bırakın.
- Koşullar döngüler ve fonksiyonlarda **{** işaretini satır sonuna koyun.
- **else** ve **else if** ifadelerinde **}** ve **{** işaretini satırın başında ve sonunda kullanın.
- İşleyicilerin başına ve sonuna birer boşluk ekleyin.

Örneğin bu kurala uygun girintilenmiş bir kod:

```
int main(string[] args){
    int i = int.parse(stdin.read_line());
    int[] liste = {12, 22, 55, 27};
    if(i == 12){
        i = 33 / 3;
    }else if(i >= 44){
        i = 0;
    }
    return 0;
}
```

Burada da aynı kodun kötü girintilenmiş hali bulunmaktadır.

```
int main(    string[] args ){
int i=int.parse(  stdin.read_line() );
int[] liste={12,22,55,27};
if (i== 2)
  {i=33/3;}
    else if    (i>=44){i=0;
        }return 0;
            }
```

Her ikisi de aslında tamamen aynı kod fakat ilk örnek daha okunaklı ve düzenli gözükmektedir.

Açıklama satırı

Açıklamalar 2 şekilde yapılır. Açıklama bölümleri derleme esnasında dikkate alınmaz.

- \ ifadesinden sonra satır sonuna kadar olan kısım açıklamadır.
- * ile başlayıp *\ ile biten yazılar açıklamadır.

```
// bu bir açıklamadır
/* bu bir açıklamadır */
```

Ekrana yazı yazdırma

Ekrana yazı yazmak için **printf** kullanılır. normal çıktı için **stdout.printf**, hata çıktısı için ise **stderr.printf** kullanılır.

```
int main(string[] args){
    stdout.printf("Merhaba Dünya");
    return 0;
}
```

Değişken türleri

Değişkenler türleri ile beraber tanımlanırlar veya **var** ifadesi kullanılarak tanımlanırlar.

```
...
int num = 0;
val text = "Hello world";
string abc = "fff";
...
```

Bununla birlikte değişkenlere başta değer atamayıp sonradan da değer atama işlemi yapılabilir.

```
...
int num;
num = 31;
...
```

Değişkenler arası tür dönüşümü işlemi için **parse** ve **to_string** kulanılır.

```
...
int num = 15;
string txt = num.to_string(); // int -> string dönüşümü
int ff = int.parse("23"); // string -> int dönüşümü
...
```

Başlıca veri türleri şunlardır:

- **int** tam sayıları tutar
- **char** tek bir karakter tutar.
- **float** virgüllü sayıları tutar.x
- **double** büyük bellek boyutu gerektiren sayıları tutar.
- **bool** doğru veya yanlış olma durumu tutar.
- **string** yazı tutar.

Diziler

Diziler birden çok eleman tutan değişkenlerdir. tanımlanırken **xxx[] yy** şeklinde tanımlanırlar.

```
...
int[] nums = {12,22,45,31,48};
stdout.printf(num[3].to_string()); // Ekrana 31 yazar.
...
```

Yukarıda **{}** kullanılarak dizi elemanları ile beraber tanımlanmıştır. Bir altındaki satırda ise dizinin 4. elemanı çekilmiştir ve string türüne çevirilip ekrana yazılmıştır. Burada 3 olarak çekilme sebebi dizilerin eleman sayılarının 0dan başlamasıdır.

Diziye aşağıdaki gibi eleman ekleyebiliriz.

```
...
    int nums = {14,44,12};
    nums += 98;
...
```

Dizinin boyutunu aşağıdaki gibi öğrenebiliriz.

```
...
    string[] msgs = {"Hello", "World"};
    int ff = msgs.length;
...
```

Vala programlama dilinde diziler basit işler için yeterli olsa da genellikle yetersiz kaldığı için **libgee** kütüphanesinden faydalanılır. Öncelikle kodun en üstüne *Using gee;* eklenir. bu sayede kütüphane içerisindeki işlevler kullanılabilir olur. Bu ifade detaylı olarak ilerleyen bölümlerde anlatılacaktır. **libgee** kullanılırken derleme işlemine *--pkg gee-0.8* eklenir. Bu sayede derlenen programa libgee kütüphanesi dahil edilir.

```
$ valac main.vala --pkg gee-0.8
```

Liste tanımlaması ve eleman ekleyip çıkarılması aşağıdaki gibidir:

```
Using gee;

void test(){
    var liste = new ArrayList<int>();
    liste.add(12);
    liste.add(18);
    liste.add(3);
    liste.remove(18);
}
...
```

Yukarıdaki örnekde **ArrayList** tanımlanmıştır. **add** ile eleman eklemesi **remove** ile eleman çıkarılması yapılır.

Listenin belirtilen index sayılı elemanı aşağıdaki gibi getirilir.

```
...
int num = liste.get(3); // 4. eleman değeri getirilir.
...
```

Listenin istenen bir elemanı aşağıdaki gibi değiştirilebilir.

```
...
liste.set(3,144); // 4. eleman değiştirilir.
...
```

Listenin eleman sayısı aşağıdaki gibi bulunur.

```
...
int boyut = liste.size;
...
```

Klavyeden değer alma

Klavyeden string türünden değer almak için **stdin.read_line()** kullanılır.

```
...
var text = stdin.read_line();
stdout.printf(text);
...
```

Koşullar

Koşul tanımlamak için **if** kullanılır. Bu ifade parametre olarak **bool** türünden değişken alır. Koşulun gerçekleşmediği durumda **else if** kullanılarak diğer koşul karşılanıyor mu diye bakılır. Hiçbiri gerçekleşmiyorsa **else** kullanılarak bu durumda yapılacaklar belirtilir.

```
...
if(koşul){
   ...
}else if(diğer-koşul){
   ...
}else{
   ...
}
...
```

Örneğin klavyeden değer alalım ve bu değerin eşit olma durumuna bakalım.

```
...
string parola = stdin.read_line();
if(parola == "abc123"){
    stdout.printf("doğru parola");
}else{
    stderr.printf("hatalı parola");
}
...
```

Koşullarda kullanılan işleyiciler ve anlamları aşağıda liste halinde verilmiştir.

Koşul işleyicileri

ifade	anlamı	örnek
>	büyüktür	121 > 12
<	küçüktür	12 < 121
==	birbirine eşittir	121 == 121
!	karşıtlık bildirir.	!(12 > 121)
&&	logic and	"fg" == "aa" && 121 > 12

||	logic or	"fg" == "aa" || 121 > 12
!=	eşit değildir	"fg" != "aa"
>=	büyük eşittir	121 >= 121
<=	küçük eşittir	12 <= 12
in	eleman içerme kontrolü	12 in {12, 121, 48, 94}

Koşullar için alternatif olarak şu şekilde de kullanım mevcuttur.

```
koşul ? durum : diğer-durum;
```

Burada **?** işaretinden sonra ilk durum **:** işaretinden sonra da gerçekleşmediği durum belirtilir.

```
...
string parola = stdin.read_line();
parola == "abc123" ? stdout.printf("Doğru parola") : stderr.printf("yanlış parola");
...
```

Döngüler

Döngüler aşağıdaki gibi tanımlanır. döngüde koşul sağlandığı sürece sürekli olarak içerisindeki kod çalıştırılır.

```
while(koşul){
    ...
}
```

Örneğin ekrana 0dan 10a kadar olan sayıları yazdıralım.

```
...
int sayi = 0;
while (sayi <=10){
    stdout.printf(sayi.to_string());
    sayi += 1; // sayi = sayi + 1 ile aynı anlama gelir.
}
...
```

Yukarıdaki örnekte **while** ifadesi sayı 10dan küçük ve eşitse çalışır. sayı 11 olduğunda bu sağlanmadığı için işlem sonlandırılır.

for ifadesi kulanılarak benzer bir döngü yapılabilir. Örneğin:

```
...
for(int i=0; i<=10; i++){ // i += 1 ile aynı anlama gelir
    stdout.printf(sayi.to_string());
}
...
```

Bu örnek while örneğindeki ile aynı işlemi gerçekleştirir.

Bir listenin tüm elemanları ile döngü oluşturmak için ise **foreach** kullanılır.

```
...
int[] i = {31, 44, 78, 84, 27};
foreach(int sayi in i){
    stdout.printf(sayi.to_string());
}
...
```

Burada **sayi** değişkeni her seferinde listenin bir sonraki elemanı olarak tanımlanır ve işleme koyulur.

Döngüden çıkmak için **break** döngünün alt satırlarının çalışmayıp sonraki koşul için başa dönülmesi için ise **continue** kullanılır.

```
...
while(true){
    int txt = stdin.read_line();
    if(txt == "abc123"){
        stdout.printf("Doğru parola");
        break;
    }else{
        stderr.printf("Hatalı parola");
        continue;
    }
    stdout.printf("test 123"); // bu satır çalıştırılmaz.
}
...
```

Fonksiyonlar ve parametreler

Vala yazarken forksiyon tanımlarız ve bu fonksiyonları parametreler ile çağırabiliriz.

```
int main(string[] args){
    write("Hello world");
    return 0;
}
void write(string message){
    stdout.printf(message);
}
```

Bir fonksiyon sadece bir kez tanımlanabilir. Fakat fonksiyonu isim olarak oluşturup daha sonra tanımlamak mümkündür.

```
...
void fff(); // isim olarak tanımlanabilir.
void fff(){
    stdout.printf("hmmm");
}
...
```

Ayrıca fonksiyonu isim olarak tanımlayıp **C** programlama dili ile yazılmış bir fonksiyon kullanabiliriz. Bu sayede kaynak kod C ve Vala karışımından oluşmuş olur. Bunun için **extern** ifadesi kullanılır.

```
// main.vala dosyası
extern void fff(string msg);
int main(string[] args){
    fff("Hello World");
}
```

```
// util.c dosyası
#include <stdio.h>
void fff(char* msg){
    fputs(msg,stdout);
}
```

Yukarıdaki örnekteki 2 dosyayı derlemek için aşağıdaki gibi bir komut kullanılmalıdır.

```
$ valac main.vala util.c
```

C kaynak kodunun gerektirdiği parametreleri **-X** kullanarak ekleyebiliriz. Bu sayede doğrudan gccye parametre eklenebilir.

```
$ valac main.vala util.c -X "-lreadline" # C ile readline kütüphanesini kullanmak için -lreadline gerekir.
```

Vala içinde C kullanabildiğimiz gibi tam tersi de mümkündür. Bunun için C tarafında fonksiyon için isim tanımlamamız yeterlidir.

```
void fff(char* message);
int main(int argc, char *argv[]){
    fff("Hello world");
}
```

```
public void fff(string message){
    print(message);
}
```

Yukarıdaki örnekte C kodu içerisinde vala ile yazılmış bir fonksiyon kullanılmıştır.

Bir fonksiyon normal şartlarda başka bir fonksiyona parametre olarak verilemez. Bu gibi durumlar için **delegate** ifadesinden yararlanılır. Önce delegate ifadesi ile fonksiyonun nasıl tanımlanacağı belirtilir daha sonra bu yeni oluşturulmuş tür parametre olarak kullanılır.

```
delegate void fff(string message);

// delegate ile kullanıma uygun fonksiyon tanımladık.
void f1(string message){
    stdout.printf(message);
}

// delegate çağırmaya yarayan fonksiyon yazdık
void f2(fff function, string message){
    function(message);
}

// main fonksiyonu
void main(string[] args){
```

```
    f2(f1,"Hello World");
}
```

Sınıf kavramı

Vala nesne yönelimli bir programlama dilidir. Bu sebeple sınıflar oluşturabiliriz. Sınıflar **Gtk** gibi arayüz programlamada kullanışlı olmaktadırlar. Sınıf oluşturmak için **class** ifadesi kullanılır.

```
public class test {
    public void write(){
        stdout.printf("test123");
    }
}
int main(string[] args){
    test t = new test();
    t.write();
}
```

Yukarıdaki örnekte sınıf tanımlanmıştır. Daha sonra bu sınıftan bir nesne türetilmiştir ve ardıntan nesneye ait fonksiyon çalıştırılmıştır.

Sınıf içerisinde bulunan bazı fonksiyonların dışarıdan erişilmesini istemiyorsanız **private**, erişilmesini istiyorsanız **public** ifadesi ile tanımlamanız gerekmektedir.

Sınıf içerisinde tanımlanmış değişkenlere ulaşmak için **this** ifadesi kullanılır.

```
...
public class test {
    private int i;
    private int j;

    public void set(int i, int j){
        this.i = i;
        this.j = j;
    }
}
...
```

Super sınıf

Bir sınıfı başka bir sınıftan türetebiliriz. Bunun için sınıf tanımlanırken *class xxx : yyy* yapısı kullanılır.

```
public class hello {
    public void write_hello(){
        stdout.printf("Hello");
    }
}
public class world : hello {
    public void write_world(){
        stdout.printf("World");
    }
    public void write(){
        write_hello();
```

```
        write_world();
    }
}
int main(){
    world w = new world();
    w.write();
    return 0;
}
```

Eğer var olan bir fonksiyonun üzerine yazmak istiyorsak **override** ifadesini kullanabiliriz.

```
...
public class hello {
    public void write(){
        stdout.printf("hello");
    }
}
public class world : hello {
    public override void write(){
        stdout.printf("world");
    }
}
...
```

Bir sınıfı birden fazla sınıfın birleşiminden türetebiliriz.

```
...
public class hello {
    public void write_hello(){
        stdout.printf("Hello");
    }
}
public class world {
    public void write_world(){
        stdout.printf("World");
    }
}
public class helloworld : hello, world {
    public void write(){
        write_hello();
        write_world();
    }

}
...
```

Signal kavramı

Valada sinyal tanımlayarak bir sınıftaki bir işlevin nasıl çalışması gerektiği ayarlanabilir. Bunun için isim olarak tanımlanan fonksiyonun başına **signal** ifadesi yerleştirilir.

```
public class test {
    public signal void sig1(int i);

```

```
    public void run(int i){
        this.sig1(i);
    }
}
int main(string[] args){
    test t1 = new test();
    t1.sig1.connect((i)=>{
        stdout.printf(i.to_string());
    });
    t1.run(31);
    return 0;
}
```

Namespace kavramı

Valada kodları alan adlarına bölerek yazmamız mümkündür. Bu sayede alan adı içine tanımladığımız fonksiyonları alan adı ile beraber çağırarak kullanabiliriz. Bunun için **namespace {}** ifadesi kullanılır.

```
namespace test {
    void print(){
        stdout.printf("Test");
    }
}
...
int main(string[] args){
    test.print();
}
```

Namespace iç içe tanımlanabilir.

```
namespace test1 {
    namespace test2 {
        void print(){
            stdout.printf("Test");
        }
    }
}
...
int main(string[] args){
    test1.test2.print();
}
```

Bir namespace alanını kaynak kodda içeri aktararak kullanmak için **using** ifadesi kullanılır. Bu ifade sayesinde belirtilen alan adındaki tüm fonksiyonlar kaynak kodda doğrudan kullanılabilir hale gelir.

```
using Gtk;

int main(string[] args){
    // İsterseniz yine de namespace adı ile kullanabiliriz.
    Gtk.init (ref args);
    // Gtk.Window yerine Window kullanabiliriz.
    var win = new Window();
```

```
    // Şununla aynı anlama gelir
    // var win = new Gtk.Window();
    ...
    // Aynı isimde var olan bir fonksiyonu namespace adı olmadan kullanmak mümkün değildir.
    Gtk.main ();
    return 0;
}
```

Sınıfları tanımlarken namespace belirterek tanımlamak mümkündür. Bunun için sınıfın adının başına namespace adını belirtmek yeterlidir.

```
public class test.cls {
    public void print(){
        stdout.printf("Test");
    }
}
...
int main(string[] args){
    var tcls = new test.cls();
    tcls.print();
    return 0;
}
```

Kütüphane oluşturma

Vala kaynak kodu kullanarak kütüphane oluşturabiliriz. Bunun için kodu aşağıdaki gibi derleyebiliriz.

```
// library.vala dosyası
public int test(){
    stdout.printf("Hello World");
    return 0;
}
```

Vala kaynak kodunu önce C koduna çevirmemiz gerekmektedir. Daha sonra gcc ile derleyebiliriz. Vala programlama dili **glib-2.0** kullanarak çalıştığı için bu kütüphaneyi derleme esnasında eklememiz gerekmektedir. Ayrıca glib-2.0 derlenirken **-fPIC** parametresine ihtiyaç duyar.

```
# Önce C koduna çevirelim
$ valac -C -H libtest.h --vapi libtest.vapi library.vala
# Sonra gcc ile derleyelim.
$ gcc library.c -o libtest.so -shared \
    `pkg-config --cflags --libs glib-2.0` -fPIC
```

Alternatif olarak aşağıdaki gibi de derleyebilirsiniz. Bu durumda C kaynak koduna çevirmeye gerek kalmadan kütüphanemiz derlenmiş olur.

```
$ valac -H libtest.h --vapi libtest.vapi \
    -o libtest.so -X -shared -X -fPIC library.vala
```

Şimdi aşağıdaki gibi bir C kodu yazalım ve kütüphanemizi orada kullanalım. Oluşturulmuş olan **library.h** dosyamızdan yararlanabiliriz.

```
// main.c dosyası
#include <libtest.h>
int main(){
    gint i = test(); // vala değişken türleri glib kütüphanesinden gelir.
    return (int) i;
}
```

Ve şimdi de C kodunu derleyemlim.

```
$ gcc -L. -I. -ltest main.c `pkg-config --cflags --libs glib-2.0` -fPIC
```

Bununla birlikte **libtest.vapi** dosyamızı kullanarak kütüphanemizi vala ile kullanmamız da mümkündür.

```
// main.vala dosyası
int main(string[] args){
    int i = test();
    return i;
}
```

Şimdi vala kodunu derleyelim.

```
$ valac --vapidir ./ main.vala --pkg libtest
```

Gobject oluşturma

Gobject kulanarak yazdığımız kütüphaneyi farklı dillerde kullanmamız mümkündür. Bunun için önce aşağıdaki gibi bir kaynak kodumuz olsun. Burada bin namespace tanımlayalım.

```
namespace hello {
    public void print(){
        stdout.printf("Hello World\n");
    }
}
```

Şimdi bu kodu aşağıdaki gibi derleyelim.

```
# Önce C koduna çevirelim ve gir dosyası oluşturalım.
$ valac -C main.vala \
    --gir=hello-1.0.gir \
    --library=hello \
    -H libhello.h
# C kodunu derleyelim.
$ gcc main.c -o main -shared \
    `pkg-config --cflags --libs glib-2.0` -fPIC
```

Burada yazdığımız nameplace alanına ait fonksiyonları ve sınıfları parametreleri ile birlikte listeleyen şablonumuz oluşmuş oldu. Şimdi bu şablondan typelib dosyası oluşturalım.

```
$ g-ir-compiler hello-1.0.gir --shared-library=libhello --output=hello-1.0.typelib
```

Son olarak dosyaları sistemimize kuralım.

```
$ install libhello.so /usr/lib/
$ install hello-1.0.typelib /usr/share/girepository-1.0/
```

Bunun yerine aşağıdaki 2 çevresel değişkeni ayarlayarak test etmemiz mümkün dür.

```
$ export GI_REPOSITORY_PATH=/home/user/gobject-ornek
$ export LD_LIBRARY_PATH=/home/user/gobject-ornek
```

Şimdi yazdığımız kütüphaneyi python ile çalıştıralım. Bunun için aşağıdaki gibi bir python kodu yazabiliriz.

```
import gi
gi.require_version('hello', '1.0')
from gi.repository import hello
hello.print()
```