

Sistem Programlama

Chapter-1 (QTranslate kurabilirsin çeviriyor)

Modern bir işletim sistemidir unix.

İşletim sistemi bir programdır, bilgisayarın kullanıcısı ve donanın arasında bir arayüz sağlar.

Bu işletim sistemi bilgisayarınızın kaynaklarını yönetir. (Dosyalar, programlar diskler, network, işlemci, ram) örn: windows, masOS, Solaris, Linux

Esnek, ayarlanabilir ve birden fazla program ve kullanıcının çalışmasına imkan sağlar.

Neden unix?

Oldukça bilimsel ve endüstriyel alanlarda çok fazla imkan sağlıyor.

Oldukça büyük sayıda ve bedava uygulamaları var.

İyi programlama ortamı sağlar.

Donanım bağımsız bir işletim sistemidir.

İnternet üzerindeki sunucu ve servisler UNIX üzerinde koşarlar. Tüm web serverlerin %65 i unix tabanlıdır.

----Early Unix History----

Ken Thompson ve Dennis Richie unix in ilk sürümlerini "Bell Labs" da geliştirmiş.

Basit ve şık bir işletim sistemi.

Diğer işletim sistemlerinin en iyi özelliklerini kendisine alarak geliştirmişler.

Programcı ve bilgisayar uzmanları için anlamı büyük.

Mini Bilgisayarlarının üzerinde dahi koşabilir.

Thompson, işletim sistemini, kendisinin tasarladığı ve "B" olarak adlandırdığı yüksek seviyeli dilde yeniden yazdı.

Dennis Richie, "B" Dili yeterli kadar efektif bir dil olmadığı için, "C" dilini geliştirmeye karar vermiş.

Daha sonra birlikte UNIX i C programlama dilinde esneklik sağlayabilmesi için yeniden "C" dili ile programlamışlar. B den daha iyi olması için.

İşletim Sisteminin küçük bir bölümü (kernel) assembly dilinde kodlanmıştır.

-----Unix Variantları/türleri-----

Unix geliştirmede 2 önemli kuruluş var, Unix System Laboratories ve BSD firmaları.

Sun firması var. Java ve Mysql teknolojilerini üreten firma.

-----Brief History Of Linux-----

Andrew Tanenbaum MINIX isminde bir yapı tasarlamış.

----Unix tabakaları----

Kullanıcı

arayüz ile çekirdeğe bağlanıyor.

Kütüphane arayüzü ile

Dosya açmak dosya kapatmak, dosya okumak vs yapıyor.

Sistem arayüz çağrıları ile process yönetimi bellek yönetimi I/O yönetimini vs yapıyor.

En alta da Donanım bölümü var.

---Unix Structure---

Unix işletim sisteminin çekirdeği kernel alt seviye fonksiyonları gerçekler(donanıma hüküm eden fonksiyonlar)

İşletim sisteminin (Kullanıcı programları da barınıyor) diğer parçaları kernele çağrıda bulunuyor (sistem çağrıları).

Shell(çekirdek) kullanıcı komutlarını kabul eder ve, komutların ürettiği verilerin görüntülenmesinden sorumludur. "örn who yazdın terminale , sonucu getirdi, o bilgileri getiriyor.ve ps aux yazdın"

200 üzerinde yararlı araç unix ile kullanıcıya sunuluyor(program ve uygulamalar). Geniş ölçüde görevleri destekler ve bu görevler dosya kopyalama, metin düzenleme, diğer mecralarda uygulama geliştirme gibi uygulamalar.

The Unix Account/ unix hesabı

Unix makinesinde girerken kullanıcı hesabına ihtiyacın var.

Kullanıcı adı ve şifreye ihtiyaç var.

Login denilen şey kullanıcı adımızdır.

Password ise şifre ekrana yansımaz. ***** olarak kalır.

Shell ekranı-----

Sisteme giriş yaptıktan sonra bazı bilgiler gözükecek ve shell prompt görülecektir. Bu arayüzden komutları girebiliriz. ismin yanında yazan & % # vs gibi. sentos da % vardır. shell simgeleri.

Shell bir programdır, Unix sisteme komut göndermek için kullanılır. Bazı komutlar tek kelimeden oluşur.

who -- sistemdeki kullanıcılar

date -- tarihi alıyoruz

ls -- bulunduğum klasördeki dosyaların listesini gösteriyor. list in kısaltılmışı ls dir.

Unix de herşey birer "Dosya" dır. (Klasörler, download klasörü)

pwd -- bulunduğumuz dizini gösterir

tab tuşuna basınca getirir, yani tamamlar mesela cat dosy "tab bas" dosya getirdi.

cat dosya adı --- text içeri shell e basar okumamızı sağlar.

---Command Syntax---

Komutlar tamamen doğru yazılmalı.

VM Ware ubuntu kuruldu.

ls -l içerik sayısı hakkında bilgiyi de getirir.

ls -a gizli olan klasörleri de getirerek içerik sayısını verir.

ls -A Gizli dosyaları gösterir (Mavi renkte olanlar gizli)

ls -F Gizli dosyalar hariç klasörleri de belli ediyor. (/ koydu)

ls -al klasoradi O klasörün detaylarını veriyor.

-----NO Shell Prompt----- chapter 1 Sayfa 17

Eğer Prompt'u görmezseniz terminal ekranında (Prompt cursor dur. neo@ubuntu:~/Ogrenci\$ gibi bişey) koştüğünüz program bitmemiştir.

Eğer programa özel bir cursor görüyorsak çıkmak gerekebilir. CTRL+Z ile program durdurulabilir.

Daha sonra tekrardan çalıştırana kadar program bekleyecektir. CTRL+C ile programı öldürebiliriz kapatabiliriz.

---Logging Out,Oturumu kapatmak--- chapter 1 Sayfa 16

Her zaman oturumu kapatın.

Shellden çıkmak için EXIT komutunu kullanabiliriz. veya ctrl+d

---UNIX File System--- Chapter 2 sayfa 2

UNIX içerisinde herşey birer "dosyadır" (programlar textler terminaller vs.)

Dizin de bir dosyadır.(Diğer dosyaları tutan bir dosya),

UNIX de sürücü harfleri yoktur. File system donanıma mantıksal bir arayüz sağlıyor ki erişebilelim

-----Working Directory----- Chapter 2 sayfa 3

pwd dir.

Şu anda çalıştığımız dizinin kendisidir. Çalışma dizini.

Absolute path basar.(Roottan itibaren gösterir bulunduğumuz yeri, en üst dizinden)

Siz başka bir dizin belirlemediğiniz sürece, komut sizin işleminizi Working Directory de gerçekleştireceğinizi varsayar.

Home dizini, bizim kişisel kullanıcı alanımızdır. -user1

cd home/kullaniciadi yerine cd -kullaniciadi kullanılabilir.

-----Unix File Hierarchy----- chapter 2 sayfa 5

Dizinler düz dosyaları ve diğer şeyleri barındırabilir.

Unix özel bir dosya uzantısını tanımaz.

---Unix Paths-- chapter 2 sayfa 6-7

Dizinler shasl ile birlerinden ayrılır.

Absolute path roottan başlayarak hedefe doğru ilerleyen yoldur.

Relative path şu anki çalışma dizininden başlar

-----Some Stangard Directories ---- chapter 2 sayfa 8

/- root directory

/dev blok ve karakter aygıt dizini

/etc host ile ilgili spesifik ayarların yapıldığı bir dizin.

/home kullanıcıların home dizinleri vardır, burada barınırlar.

/lib çeşitli diller için kütüphane dizini.

//sbin sistem için

/tmp geçici dosyalar

/usr kullanıcı programları uygulamalarının bulunduğu

/var sistem dosyaları burada tutuluyor. (loglar, emailler)

----Changing Directories----- chapter 2 sayfa 9

cd çalışma dizininizi değiştirir

absoulute veya relative path kullanabiliriz.

cd hiç bir argüman olmadan da tilda ile kullanılabilir direk root'a gider.

Eğer önceki yıllarda üretilen bir dosya varsa ls sorgularında saat bilgisi gelmiyor, sene bilgisi geliyor 2017 2018 vs. ÖNEMLİ!!!!!!!!!!!!!! Standart dizinler için 4096 standart bir değerdir boyut.

LS Çıktılarında

1. Kısımdaki ilk karakter dosyanın tipini verir. d olan dizin - olan düz text dosyası gibi bişey. l de link dir.

1.Kısımın son 9 karakteri nin ilk 3 ü kullanıcıya, ikinci 3 ü gruplara, üçüncü 3 lü ise dışarıya olan izinlerle ilgili. -rwrwrwrwr gibi

ls çıktısındaki "total", blok altındaki dosyaların toplam diskte topladığı blok sayısını veriyor.

-----Dosya tipleri sayfa 11-----

Binary(pdf dosyası,resim dosyası, exe dosyası, word dosyası) veya text ler Plain(-) text

Directory (d) gerçek dosyadır, başka bir dosya seti kümesini içerisinde barındırır.

Link(l) Bir başka dizine işaretçi tutuyor.

Special da b diskler veya roylar, c için ise karakter dir keyword veya joystick.

“-” işareti ile gösterilenler normal dosyalardır.(.txt, .log, .cfg, .ini vb)

“d” harfi ile gösterilenler dizinlerdir.

“c” harfi ile gösterilenler “character device file” olarak adlandırılan dosyalardır. Bu dosyalar donanım sürücülerine erişim sağlayan basit arayüzlerdir. Burada arayüz kelimesinden grafik arayüz anlaşılmasın. Belki “arayüzlerdir” demek yerine “köprülerdir” demek daha doğru olabilir. Örneğin yazıcılar, tarayıcılar gibi çevre birimlerine erişimde kullanılır.

“b” harfi ile gösterilenler “block device file” olarak adlandırılan dosyalardır. “character device file” olarak adlandırılan dosyalarla benzerlik gösterirler. Harddisk, ram gibi birimlere erişimde kullanılırlar.

“s” harfi ile gösterilenler “local socket file” olarak adlandırılan dosyalardır. Process’ler arasında haberleşmek için kullanılırlar. Genelde X windows, syslog gibi servisler tarafından kullanılır.

“p” harfi ile gösterilenler “named pipe file” olarak adlandırılan dosyalardır. Socket dosyaları gibi bunlarda process’ler arasında veri gönderilirken kullanılır.

“l” harfi ile gösterilenler “symbolic link” olarak adlandırılan dosyalardır. Bu dosyalar sayesinde bir sistem yöneticisi bir dosyaya işaret eden birden fazla dosya üretebilir.

“D” harfi ile gösterilenler “door file” olarak adlandırılan dosyalardır. Bu dosyalar bir istemci ile sunucu arasındaki haberleşmede kullanılan özel dosyalardır.

ÖDEVVVVVV başka harfler de var araştır!

---Manipulating Files--- sayfa 12

touch yeni bir dosya oluşturmaya veya var olan dosyanın son modifiye tarihini değiştirmeye yarar.

touch dosyaadi ----- dosya oluşturur.

mv komutu taşımaya yarıyor.

mv ubuntu windows ----- ubuntu isimli dosyanın adını windows yaptı.

mv ubuntu ../ bir üst dizine taşıdı, veya uzun dizin de verebiliriz başka yola taşımak için.

cp kopyalıyor

rm siliyor.

mkdir dizin yani klasör oluşturuyor.

rmdir dizini siler.

man yardım komutu

rm-r kompçe uçurur herşeyi

ln -s sembolik link oluşturur. windowstaki kısayol gibi.

omit -s aynı cihazın aynı fiziksel partitionunda olmak zorundadır. hardlink oluşturur.

-----ÖDEVLER-----

hardlink araştırma yap?

Hardlink doğrudan dosyayı gösteren adrestir. Dosyalar fiziksel bir nümerik adreste bulunuyor. Linux seviyesinde bu inode numarası oluyor. Hard linki aynı olan dosyaların bu numarası da aynı oluyor.

Special da b diskler veya roylar, c için ise karakter dir keyword veya joystick.

ÖDEVVVVVV başka harfler de var araştır!

“-” işareti ile gösterilenler normal dosyalardır.(.txt, .log, .cfg, .ini vb)

“d” harfi ile gösterilenler dizinlerdir.

“c” harfi ile gösterilenler “character device file” olarak adlandırılan dosyalardır. Bu dosyalar donanım sürücülerine erişim sağlayan basit arayüzlerdir. Burada arayüz kelimesinden grafik arayüz anlaşılmasın. Belki “arayüzlerdir” demek yerine “köprülerdir” demek daha doğru olabilir. Örneğin yazıcılar, tarayıcılar gibi çevre birimlerine erişmede kullanılır.

“b” harfi ile gösterilenler “block device file” olarak adlandırılan dosyalardır. “character device file” olarak adlandırılan dosyalarla benzerlik gösterirler. Harddisk, ram gibi birimlere erişmede kullanılırlar.

“s” harfi ile gösterilenler “local socket file” olarak adlandırılan dosyalardır. Process’ler arasında haberleşmek için kullanılırlar. Genelde X windows, syslog gibi servisler tarafından kullanılır.

“p” harfi ile gösterilenler “named pipe file” olarak adlandırılan dosyalardır. Socket dosyaları gibi bunlarda process’ler arasında veri gönderilirken kullanılır.

“l” harfi ile gösterilenler “symbolic link” olarak adlandırılan dosyalardır. Bu dosyalar sayesinde bir sistem yöneticisi bir dosyaya işaret eden birden fazla dosya üretebilir.

“D” harfi ile gösterilenler “door file” olarak adlandırılan dosyalardır. Bu dosyalar bir istemci ile sunucu arasındaki haberleşmede kullanılan özel dosyalardır.

Her bir dosyanın tek bir sahibi vardır.

--chown komutu ile sahibi değiştirilebilir.

Yalnızca root bu komutu kullanabilir.

yalnızca 1 dosya 1 gruba dahildir.

u User g Group o World (ugo kelimesi) file permissionlar.

--man chown kodunu dene. (man o kod hakkında yardımları getiriyordu.)

"sudo su" komutu ile login yaptı.

"chown root selam" yazarak rütbeyi aldı.

"exit" yazarak root kullanıcısından çıktı.

tekrar ls ile listeleme yaptığıında selam dosyasının sahibi root olmuş oldu. listede gördü onu sende dene.

-----File Permissions ---- sf 16

3 farklı izin türü var. Sahibinin izinleri, sahibinin bulunduğu grubun izinleri, bunların haricinde dış dünyaya verilecek izinler. (u-g-o simgeleri user group ve dış kullanıcılar)

----sf 17-----

chmod 777 textfile tam yetki verir.

touch yirmiekim (yirmiekim diye bir dosya oluşturdu)

chmod 700

chmod 755 --- standard genelde verilen izin budur webde.

chmod 444 tam güvenlik (sql injection engellemek için gerekebilir.) Hiç kimse yazma yapamıyor.

chmdod g+rw textfile -->>> bu kodu dene.

-----File Modification date ---- sayfa 18

touch komutu da dosyanın tarihini şu anki tarihe güncellemek amacı ile kullanılabilir

Eğer dosya yoksa touch yeni dosya oluşturur, yoksa tarih değişiyor.

---sf 19---

cat concatenate in kısaltılmıştır.

cat komutu ile dosyanın içeriğini görebiliriz.

cat textfile1 textfile2

less/more textfile

(dosya ismi boyutuna göre boşluk koyan kod)

---wildcards in file names ---- sf 20 REG EXP! ÖNEMLİ / bunlarla baya bir işimiz olacak.

*herhangi bir string için gelebilir veya gelmeyebilir.

? herhangi bir karakter için. doc? tek bir karakter getiriyor veya getirmiyor.

[] bir karakter aralığını belirtir (a-c) ad an c ye kadar herhangi bir karakter. a veya b veya c getirir. [a-c]* için yıldız olduğu için boş string bile olabilir. Kişinin isim yazıp yazmadığını kontrol edin? Teşekfon numarasını mı girmiş? Mailini düzgün girmiş mi?

---Getting Help on Unix commands--- sf 21

man komutu ile komut ile ilgili tüm dokümantasyonları getirir.

apropos kelime kodu yanda belirttiğimiz keyword u içeren tüm konumları getirir. örn apropos list

locate string komutu stringi içeren tam yoldaki tüm dosyaları gösteriyor

örn locate home/neo

locate home/neo > cikti kodu büyük işaretinden sonra verilen dosya ismine aktarılır. tüm dosyaları basıyor bu şekilde yaparsak, yolda olan olmayan.

--Other file systems- sf25

tmpfs

specfs -- özel karakter ve blok aygıtlarına erişim sağlar.

lofs sanal dosya sistemleri oluşturuyor ki var olan dosyaların üzerine yazabilir.

tfs var olan dosyaların üzerine bir dosya sistemi kurulmasını sağlıyor.

fdfs işaretçiler yardımı ile dosya içeriğine erişim sağlar

namefs streams ları kullanarak dosyalar üzerindeki dosya işaretçilerini birleştiriyor.

swapfs swap alanının kernel alanının yönetilmesi için kullanılıyor.

(Buralardan bir kaç tanesi eksik kaldı chapter 2 den çevir üst tarafı.)

CHAPTER-3

---TEXT Editing--- sf 2

her önümüze gelen dosyayı editör ile açıp değiştiremeyiz.

cat ve less kullanarak editör açabiliriz, text editörler için.

Neden vi? sf 3

Komutlar aynı zamanda anahtarlardır.

mouse olmadığında bunu kullanmak zorunda kalacağız.

basit ve güçlü bir metin editörü.

Vi Improved i (vim) olarak adlandırabiliriz.

-----vibasics----- sf 4

vi [dosyaadi] veya adları, hepsini açabilir editörde bu komut ile.

Tuşlar komutlardır.

insert modunda karakterler eklemek için texti düzenlemek için kullanılabilir.

escape karakteri de mevcuttur. yani insert modundan tekrar komut moduna dönebilmek için kullanılacak esc.

-----Command Mode----- sf 5

shift ile iki defa z bas komut modundan çıkabilirsin.

-----Command Mode----- sf6

I/O REDIRECTION

-----Three Standard Files----- sf 2

stdin terminalden default giriş alıyor.

stdout terminale default çıkış veriyor.

stderr hata mesajlarının alınmasını sağlayan default terminale çıkış yapıyor.

-----Redirection stdout ----- sf 3

çıkıyı > ile bir dosyaya yönlendirebiliyoruz.

örn stdout > dosyaadi (önceden eklenenleri siler.)

>> ise dosyaya çıkıyı eklemeye yarıyor. (önceden eklenenleri silmez.)

echo \$PWD sistem değişkenini çıkıya bastı.

cat selam >> ubuntu selam ı sonra bastı dosyada.

cat ubuntu

< işareti ile bir kaynaktan veri okuyabiliyoruz.

mail user@domain.com < message.txt (message.txt den mail içeriğini aldı ve attı)

sort < friends > sorted_friends (friends kaynağından aldı sıraladı ve sorted_friends bastı)

sort sayılar sayıları gitti sıraladı.

sort -n sayılar yazarsak -n sayısal olarak sıralaması gerektiğini anlıyor string değil.

sort -n sayılar > siralanmis_sayilar sayilari siraladi sağa bastı.

sort -n < sayilar> siralanmis_sayilar

----Filters----- sf 5

wc argünel olarak verdiğimiz satır ve karakterler hakkında bilgi veriyor.

L w ve c önemli parametreler -L -l -w -c

wc siralanmis_sayilar 9 satır var 9 kelime var 27 tane karakter var.

neden wc çıktısı arayüzdeki çıktıdan farklı veriyor. (arayüz: Document Statistics diye bir sayfa)

sort -nr < sayilar > siralanmissayilarim (tersten sıralıyor sanırım)

sort < kelime-listesi > sıralanmisliste küçüktn büyüğe sıralıyor.

cut -cl-5 yirmiekim (dosyanın karakter modunda 1. karakterinden itibaren 5. karakterine itibaren olan kısmını al, her bir satırda yapıyor bu işlemi)

cut -cl,4 ilk paramtre ile son parametreyi alıyor virgül koyarsak. 1. ve 4. karakterleri alarak 2 karakter gösteriyor.

cut -d: -f1,5 /etc/passwd "d"delimiter in kısaltmış, ayraç demek. bu komut iki noktaya göre metni parçalıyor. "f" file anlamına geliyor. 1. ve 5. elemanları alıyor

----Filters----sf6

head -6 yirmiekim baştan itibaren 6 satırı alıyor.

tail -6 yirmiekim sondan itibaren 6 satır alıyor.

sort +2n ve tail +n çalışmıyor.

diff yirmiekim yirmiekimek (küçüktür işaretleri)

od -c yirmiekim oktal olarak gösteriyor verileri

ls -lt dosyaları oluştukları zamana göre sıralıyor.

NOT: Head tail ls vs önemli iyi öğren.

crypt şifreleme yapar, ama programı kurmak gerekiyor. önemli değil.

-----Pipes----- sf7

pipe bir komutun çıktısını bir başka komutun girdisine bağlamak için kullanılır.

ls -la | less

ls -la | wc

ls -al > lscikti

cat yirmiekim | wc

man bash | grep "history"

ps aux | grep user1 | wc -l çıktı üzerinde "user1" i ara, onun satırlarını getir, sistemdeki processlerin sayısını getiriyor.

ps sadece kullanıcı processlerini gösterir

ps aux ise sistem processlerini de gösteriyor

who | sort > anlkkullanıcılar

----Processes--- sf8

date tarih getiriyor

date ; who araya noktalı virgül koyunca processler ardı ardına icra ediliyorlar.

ÖNEMLİ!

ls - al & wc * (yıldız, terminaldeyken, bulunduğunuz yerdeki dosyalara karşılık geliyor)

----More Commands: Communication-- sf9

talk aynı makinadaki kullanıcılar arasında konuşma sağlar

write de aynı şekilde mesaj yollamayı sağlıyor.

mail -text- mail sallıyor.

ftp -text-based FTP program

lynx -text-tabanlı web sayfası. yine programı kurmak lazım.

-----More Commands sf 10..-----

top o anki sistem ile ilgili bilgileri veriyor bize. (Uyuyan zamanları saymıyor) yük değerleri çok önemli. Sistem hakkında bilgi veriyor bize.

Tasks: process sayısı. Kib Mem: ram miktarı., free: boş olan kısım. active memory aktif olan kullanım.

ÖDEV: Zombie process nedir?

vmstat -s komutu ile gerçek anlamdaki kullanım oranlarını alabiliriz. (topdaki ram kullanımı çok tutarlı değil cache falan da katıyor gibi birşey var ortada emin değilim.)

kill process öldürüyor, hatta oturum kapatabilir(araştır.)

time ps de ps komutunun ne zamandır çalıştığını gösteriyor

---Unix programs that use REs---sf2

sed ve awk önemli.

grep

----Basic vs Extended REs ----sf4

egrep pattern filename (Patternin işleneceği dosya ismi.)

Emin olmak için patterni çift tırnak içine alabilirsiniz.

egrep "abc" dosyaadi

egrep "oyun" kelime-listesi

egrep -i "abc" dosyaadi büyük küçük harf önemsemeden yapıyor arama işlemini.

egrep -v "a" dosyaadi içermeyenleri getiriyor.

egrep -n "a" dosyaadi hangi satırlarda olduğunu da getiriyor.

egrep -ni "a" dosyaadi büyük küçük önemsemeden getirdi satır numaraları ile.

---Metacharacters-- sf 5

(.) herhangi bir karakter ile eşleşir. nokta yerine herhangi bi karakter gelebiliyor.

egrep -ni "f..a" kelime-listesi f ile başlayıp a ile biten kelimeleri getirdi.

(*) hiç birşey olamayabilir veya bir önceki gelenden istenilen kadar gelebilir.

"ab*c" ac abc abbc abbbc abbbbc gibi bişey getirebilir. birsürü veya hiç tane.

".*" herhangi bir karakterden hiç gelmeyebilir sonsuz gelebilir.

"n.*" n ile başlayan herşeyi getiriyor.

NOT:Shellde kullanılan yıldız bulunulan yerdeki dosyaları belirtir. Düzenli ifadede kullanılan yıldız ise ya hiç yada istenilen sayıda anlamına gelir.

-----Metacharacters ----- sf 6

(+) en az 1 tane yada istenilen kadar getirir.

"ab+c" abc abbc getirir ama ac getiremez.

(?) ya 1 tane getirecek yada hiç getirmeyecek.

"ab?c" abc tanır ama abbc tanımaz.

Logical (|) mantıksal olarak herhangi birine uyunca onu kullanır. normal veya gibi bişey yani.

"abc |def" abc veya def ile eşleşir.

----Meta Characters---- sf 7

Caret ^D.* D ile başlayan herşeyi getiriyor. üssü işareti satırın başında arama yaparken kullanabileceğimiz bir karakter.

Dollar sign \$ satır sonunda arama yaparken kullanılabilir .*d\$ "d" harfi ile biten düzenli ifadeler.

Backslash \ diğer meta karakterlere escape yapıyor. file\.txt file.txt yi getirir. file_txt yi getiremez.

--Meta characters--- sf 8

[fF]un fun veya Fun u getirir.

b[aeiou]g bag beg big bog bug getirir.

[A-Z].* Büyük harf ile başlayan herhangi bir satır ile eşleşir. tire işareti aralık belirtir.

[^abc] bunlarla başlayamaz demek.

-----Metacharacters----- sf9

a(bc)* a, abc, abcbc, abcbcbcb eşleşir

(foot|basket)ball football veya basketball eşleşir

süslü parantezler kaç tane gelmesi gerektiğini yazar yani;

[a-z]{3} 3 tane küçük harf ile eşleşir.

m.{2,4} ilk parametre en az olabilecek sayı iken 2. karakter en çok olabilecek sayı. yani en az 2 en çok 4 getirebilir. m harfi ile başlar sonrasında . yerine herşey gelebilir en az 2 en fazla 4 tane.

---What do these mean?--- sf 10

egrep "^B.*s\$" Büyük B ile başlayabilir, araya birsürü şey gelebilir veya gelmeyebilir. s ile biter.

egrep [0-9]{3} 0 ve 9 arasındaki rakamlardan 3 basamaklı sayı getirir var ise.

976873 yazdım ne yapar?

egrep num(ber)? [0-9]+ num veya number durumunu kontrol etmiş.

egrep "word" file | wc -l word kelimesini getirecek -l olduğu için Lines satır sayısını getirecek word içeren.

egrep [A-Z].*\? sonda gerçekten soru işareti bekliyor çünkü / ile escape atmış. büyük harfli bir soru cümlesi bekliyor.

ls -l | egrep "^....r.-r.-" yetkisi bu olanı getirdi. gruba okuma izni verilmiş ÖNEMLİ!

egrep "^[^:]*:/" /etc/passwd ÇOK ÖNEMLİ! ŞİFRESİ OLMAYAN KULLANICILARI GETİRİYOR

^[^aeiou]*a[^aeiou]*e[^aeiou]*i[^aeiou]*o[^aeious]*u[^aeiou*]*\$ küçük SESLİ harflerde aeiou hepsini içeren ve alfabetik sırada içerenleri arıyor.

ÖDEV:

egrep -f reguler kelime-listesi alfabetik kelimeleri alıyor, kelime-listesi ne yazıyor.

mail reguler expression. alan adlarının expression u.

^[a-zA-Z0-9]\+@[a-zA-Z0-9]\+\. [a-z]\{2,\}

satır bir kelime ile başlayacak ve o kelime en az 10 harf ile başlayacak.

/^[a-zA-Z]{10,}\$/

Satırların içerisinde bir öğrenci numarası olsun ama 3 kısımdan oluşan bir numara olsun.

Ardışık 2 büyük harf gelen satırlarının sayısı

([A-Z]){2}

alfabetik karakter ile bitmeyenlerin satırlarının sayısı

Cümle sonunda ki kelimedede sesli harf ile başlayan satırları getir.

--sf2--

Pencere sistemleri grafiksel arayüz sağlıyormuş pencereler ikonlar sayesinde.

Kullanılabilirliği artırıyor (kullanışlıdır) çoklu araçlara erişim konusunda

ve uygulamalara erişim konusunda.

Fare joystick tabletler gibi şeylerle direk yönetim sağlayabiliyoruz sistem üzerinde.

--sf3--

Unix pencere sistemlerinden önce komut satırları ile çalışmak üzere tasarlanmış onun üzerine diğer yenilikler geliştirilmiş.

Modern Unix sistemleri pencere sistemlerini destekliyor ve pencere sistemlerinin avantajlarını kullanmaya olanak sağlıyor aynı zamanda profesyonel kullanım için komut satırlarının kullanımına da izin veriyor.(bu siz ineklerisiniz!)

--sf4--

Bütün UNIX sistemleri X Windows (XFree86) temeline inşa edilmiş.

Standart Version: X11R6

X Server:

Donanım Arayüzü (ekran, fare, vs.)

Ekran boşluklarını yönetiyor

Basit grafikler çiziyor

X Clientlere pencere tanımlamaları sağlıyor.

Yerel ve uzaktan clientleri destekliyor.

--sf5--

x-Client server mimarisi

X ağ üzerinde çalışmak üzere tasarlanmış

X server: Programın çıktısının gösterileceği makina üzerinde koştan yazılımın kendisi.

X client: Aynı veya başka bir makinada koştan client.

Client çizme ve diğer Xclient komutlarını gönderiyor sonuçları gösteren sunucuya.

--sf6--

X'in tarihsel gösterimi:

Kullanıcılar XTerminal de Sat? yapar – grafiksel terminaller X server'de koşabilir.

, ama işletim sisteminde koşamazlar (OS)

Uzak bir UNIX makinasına login olmuş kullanıcılar veya diğer işletim sistemi (OS) destekleyen

X Clientler grafik arayüzler ile uygulamaları yönetebilirler, kombine bir şekilde

birden fazla bilgisayar üzerinde çalışan uygulamaları koşabilirler.

--sf7--

X'in getirdiği yenilikler:

Transparent remote execution??

Her bir programa kendilerine ait görsel ekranlar veriyor.

Önemli pencereleme işlemlerini de içerisinde kapsıyor.

Window damage---Pencere hasarı?

Window reveal events ??

Backing store Arka dükkan??

X11 programları yüksek derecede taşınabilirdir.

--sf8--

Pencere yöneticisi:

Pencere yöneticisi X11 in en üstünde koşar masaüstü yöneticisi ile birlikte.

Pencereler üzerindeki Kenarlıklar, Sliderlar, ve diğer çıkartmaları arayüzde daha görülebilir ve hissedilebilir olmasını sağlar.

Örnekler:

kwin - KDE için varsayılan.

metacity – -GNOME için varsayılan.

mwm – Motif bağımsız pencere yöneticisi.

--sf9--

Kullanıcı Arayüzü (Masaüstü Ortamı)

Kullanıcı arayüzü, Dosya yöneticilerini, ikonları, panelleri, konfigirasyona araçlarını ve appletleri organize ederek arayüze entegre ediyor ve görmemizi sağlıyor.

GNOME (GNU)

GIMP ile geliştirilmiş Toolkit (GTK+)

KDE (kullanıyor) Qt C++ kütüphanelerini

Xfce (GTK+ tabanlı)

Unix in çevre değişkenleri ve çevre ortamını inceleyeceğiz bu chaapterde.

-----Shell Characteristics-----sf2

Komut satırı işletim sistemi ile kullanıcı arasında bir arayüzdür.

Hem bir komut arayüzü hemde bir programlama dilidir.

Shell script bir text dosyasıdır.

-----Shell Interactivity-----sf3

Kelimesel tamamlama

takma adlar

komut satırı düzenleme

komut geçmişi

ayarlama.

-----Shell Programming-----sf4

Değişkenler var

Kontrol yapıları var.(döngüler ve şart durumları yani if else vs)

Fonksion yanımlamaları ve çağırma

Shell scriptler.

-----Various Unix Shells-----sf5

sh(bourne shell,priginal unix shell)

ksh (kornshell)

csh c shell, developed at berkeley

tcsh

bash bourne again shell (bizim göreceğimiz shell budur.)

bash sh nin gelişmiş halidir.

Unix default shell i Bash dir.

-----Bourne again shell (bash)-----sf6

standart kabuktur.

diğer shell türevlerinden bazı özellikleri de kendisine katmıştır.

-----Environment Variables (çevre değişkenleri)-----sf7

Değişkenler genel anlamda isim ve değeri vardır.

isim=ali

echo \$isim ----- çıktı ali olur. direk terminalde deneyebilirsin.

env tüm çevre değişkenlerini getirir.

HOME,PATH,USER Önemli

-----Environment Variables Examples -----sf8

env > envcikti

grep HOME envcikti

grep PATH envcikti

grep ^PATH envcikti

grep HOME envcikti

grep PWD envcikti

grep PS1 envcikti ---- gelmedi

grep HOSTNAME envcikti ---- gelmedi

set > setcikti

grep PS1 setcikti ---- geldi

grep HOSTNAME setcikti ---- geldi

bunun sebebi bu 2 değişken env içerisinde bulunmuyor ama yine de çevre değişkeni.

export komutu ile ARTIK env listesi içerisine girecektir yani şöyle; "export PS1"

echo \$HOME ana dizini gösterir

echo \$PWD ise bulunduğumuz yolu gösterir

echo \$PS1 bir kullanıcıdır, aktif olarak kullanmayacağız

En önemliler PATH,

echo \$PATH

PATH = \$path: \$HOME/BİN

Program içinde tarihi alabilmek için;

ali=`date` (altgr virgül ile açtı tırnağı)

echo \$ali

veli=\$(date)

echo veli

yer = \$(pwd)

yer2= `pwd`

echo yer

echo yer2

pu yaz ve tab bas, tamamlıyor pushd gibi yani veya purple-blala vs.

-----Alias-----sf11 (takma adlar)

alias liste = "ls -al"

sonra direk liste yaz enter bas bu kodu kosacak.

alias tarih=date

tarih yaz enter bas.

history ise önceden kullandığımız komutları getirir.

fc -l 1990 1995 dersek soldaki sıra numarası arasındaki komutları getirir.

-----Command line ending-----sf13

ESC B eski kodu getirdi

ESC F kelimelerde gezdi

CTRL K cursor sonrasını siliyor

-----Login Scripts-----sf14

-----Login Scripts-----sf18

ENV= \$HOME/.cshrc

CAT .bashrc ÖDEV

Ubuntu üzerinde login script yap. Basit bir alias koyalım mesela dizini listele. mesela sirala diyince ls -al gibi. ama biz login script yapacağız.

-----Shell Execution -----sf4

bash my_script koşturmak için bu komut kullanılabilir.

-----Numeric Variables -----sf7

bash my_script koşturmak için bu komut kullanılabilir.

var+=1 değişkene 1 ekler

var++ artırıyor

var2=1+\$var var değişkenine 1 ekleyip var 2 atıyor.

-----String Variables ----- sf 8

farklı bir türde declare etmedikçe tüm değerler stringdir.

İki parantez içerisine koyarsak integer olarak işlem yapabilir.

((var2=1+\$var)) gibi.

-----String Variables ----- sf 9

\${string:5} ilk 5 karakteri atla sonrasını al demek.

\${string: -2} son 2 karakteri alıyor.

\${string:2:10} 1. rakam, konumu, 2. rakam o komuttan sonra kaç karakter geleceği.

\${#string} string karakterin boyutunu veriyor. length.

ARRAY VARIABLES SF10-----

Referans işlemi \${name[index]}

\${a[3]} 4. elemana karşılık gelir.

declare -a ile tanımlama yapılır.

declare -a sports boş dizi tanımladı

sports=(basketbol futbol tenis kayak) içine verileri attı.

echo \${array[*]} tüm dizi içeriğini bastırır.

-----Export Variables ----- sf12

export edince her tarafa erişim açabiliyoruz.

-----Komut satırı argümanları sf13

Bir scriptte argümanlara geçtiğimiz zaman \$1 \$2 \$3 ----- \$9 a kadar direk erişim sağlayabiliyoruz.

\$0 scriptin ismidir.

\$* argümanları bir string şeklinde döndürür.

\$@ benzer şekilde argümanları döndürüyor ama bir dizi şeklinde dönderiyor.

\$#

----- sf 14

echo -n "yes/no? " soru yolluyor kullanıcıya

tek tırnak ile yazarsa olduğu gibi basar yorumlamaz.

echo " 'date +%D' "

-----Shell Variables sf 17

echo "isim gir"

read name

kullanıcının girdiğini name değişkenine alıyor.

değişken tanımlarken içeriği boşluk bırakarak atayamazsın.

mesela FRUIT= apple orange plum hatalı.

FRUIT= "apple orange plum" doğru.

cat read_name

echo Please enter your name"

read name

echoWelcome to CE dept. KTU, \$name"

-----Computation on Variables sf21

echo 'expr \$a + \$b' toplama işlemi yaouyor

expr 5 + 7 çıktı 12 olur

sf 23-----

\$1 - \$9 pozisyonel parametrelere direk erişim sağlıyor.

\$0 scriptin adıdır.

\$# argüman sayısını veriyor.

\$? koştan son komutun çıktısını vermek için kullanılır.

\$\$ PROCESSİN ID numarasını veriyor.(bash olarak koştığımız text dosyası bile bir process)

#! bu arka planda koştan en son komuta ait process ID yi veriyor.

\$@ argümanları dizi şeklinde alıyordu (bash emre 10 20 30 bu parametreleri basıyor)

\$@@ ise \$* gibi işlem yapar.

egrep ile reguler ifadeleri kullanıyoruz.

sf25-----

ps -aux tüm processleri gösterir. Bash text adı olarak koştığımız processi burada görebiliriz.

(eğer sleep var ise.)

sleep 50 & echo 'hi there' sleep program arka planda koştaya devam ediyor aslında ve durdurmuyor programı. ÖNEMLİ!

sf 28-----

Eğer 9 parametreden fazla parametre script'e aktarılırsa, 9 un üzerindeki parametrelere DOĞRUDAN erişim OLAMAZ. bunun sebebi \$dan sonra yalnızca 1 tane rakam kabul etmesidir.

shift komutu 1 pozisyon sağa kaymasını sağlar. bu sayede 2 rakamlı parametrelere de erişebiliriz.

while [\$# -gt 0] parametre sıfırdan büyük ise while işlemine giriyor

sf 31-----

&& bir önceki komut düzgün bir şekilde çalışmış ise sonraki komutu icra eder.

|| sol taraf başarısız olur ise sonraki komutu koşturur.

sf 34-----

\$? koştan son komutun çıktı durumunu öğrenmek için, yani yalnızca okunabilir değişken olan \$? değişkeni içerisinde tutulur.

Eğer \$? ın değeri 0 ise başarılı çalıştı demektir. farklı birşey ise başarılı çalışmadı demektir.

sf 35-----

if kontrollerinde;

integerlar ile birlikte ((condition))

stringler ile birlikte [[condition.]]

string karşılaştırılmasında tek = kullanılır [[emre = emre]] gibi.

[[-e \$file]] if içerisinde bi dosya var mı yok mu onu kontrol ediyor. (SINAVDA ÇIKACAK %100)

[[-f \$file]] düzenli dosya olup olmadığını kontrol ediyor. (SINAVDA ÇIKACAK %100)

[[-d \$file]] bir dizin olup olmadığını kontrol ediyor.

[[-L \$file]] sembolik link mi değil mi?

[[-r \$file]] OKUNABİLİR Mİ?

[[-w \$file]] yazılabilir Mİ?

[[-p \$file]] pipe Mİ?

test komutu herhangi bir çıktı üretmez ama test komutunun çıktı durumu if yapısı şeklindeki bir kontrole başarılı olup olmadığını aktarabilir.

echo \$? ile kontrol sağlanıyor yani.

-z bir string boş mu değil mi? zerodan geliyor

-n de yine string uzunluğunu kontrol ediyor, sıfır mı değil mi length?

name ="Ahmad"

test -z \$name 1 yani false değeri dönecek

test -n \$name 0 döner

sf 47----- integer test komutları

-f REGULER(DÜZENLİ) dosya var mı yok mu testi için kullanılır.

-s ile dosya var mı yokmu kontrol ediyor, birde boyutu 0 mı değil bi

-d izin mi diye kontrol eder.

-r okuma izni var mı

-w yazma izni var mı

-x çalıştırılabilir izni var mı

ÖDEV: 7.SLAYT SF 14 DEN LOGİN SCRİPTE BAKABİLİRİZ.

SINAV FOR BİTMİŞ WHILE DAHİL, UNTIL LOOPS 8. CHAPTER SF 59 SON.

ÖDEV:DFS DEEP FİRST SEARCH algoritması ödev olarak verecek ileride!

9. hafta 1. ders

-a : AND

-o : OR

! : NOT

ÖRNEK: test -r "mydoc.doc" -a -w "mydoc.doc"

mydoc.doc dosyasını yazma ve okuma izni var mı?

----- Case (çift ; kullanılır ;;)

CELAL 2 dosyası;

clear

echo "1. Date and time"

echo

echo "2. Directory listing"

ccho

echo "3. Users information"

echo

echo "4. Current Directory"

echo

echo "Enter choice (1, 2,3 or 4):"

read choice

case choice in

1) date;;

2) ls -l;;

3) who ;;

4) pwd ; ;

*) echo wrong choice ;;

esac

[#for](#) colors in s@

for colors in Red Blue Green Yellow Orange Black Gray White

do

echo Sclor3

done

-----For loop

celal 3 dosyası:

```
for i in $*      // parametreler string olarak verilir. Renkler ile aynı durum geçerli. Ayraç olarak boşluk kabul edilir.
do              // forun başlangıcı
    if who | grep -s $i > /dev/null    // who komutu üzerinden parametre olarak verilen değişken aranıyor ($*) boş çıktı
    then
        echo $i is logged in // true döner
    else
        echo $i not available // false döner
    fi
done
```

bash celal3 ogrenci root celal çalışır. teker teker sonuç döner.

-----While loop

----- örnek 1

#!/bin/bash

while echo "Please enter command" // komut girilir.

read response // girilen komut okunur.

do // döngü başlar

case "\$response" in

'done') break ;; // eğer done yazılmışsa işlem biter

"" continue ;; // döngü başa döner

*) eval \$response ;; // başka bir komut girilirse eval ile çalıştırılır.

esac // case biter

done

----- örnek 2

echo What kind of tree bears acorns \? // meşe palamudu ne taşır?

read response // değer girilir ve girilen değer responsa atılır.

case \$response in // değer kontrol edilir

[Oo][Aa][Kk]) echo \$response is correct ;; // oak için tüm durumlar kabul edilir.

*) echo Sorry, response is wrong // diğer durumlarda yanlış cevap.

esac // echo biter

----- örnek 3

```
clear
echo What is the Capital of Saudi Arabia \? // arabistanın başkenti?
read answer // cevap okunur
while test $answer != Riyadh // test içine aldığı değerden 0 ya da 1 döndürür. (0dan büyük olanlar hep true döner.
do
    echo No, Wrong please try again. // riyyadh değilse 1 döner (!) cevap yanlıştır.
    read answer
done
echo This is correct. // riyyadh ise doğrudur.
```

----- örnek 4

```
clear
echo "Please Enter the user login name: \n" // isim girilir
read login_name // login_name ye atılır
until who | grep $login_name // burası doğru olana kadar döner.
do
    sleep 5
    echo "Wrong name! Please try again: \n" // yanlış olduğu sürece tekrar giriş bekler.
    read login_name // tekrar giriş bekler
done
echo The user $login_name has logged in // doğru olduğunda çıktı gelecek.
```

----- Select loop

```
select name in word1 ... wordN // değişken ismi ve değişkenler.
do
    list
done
```

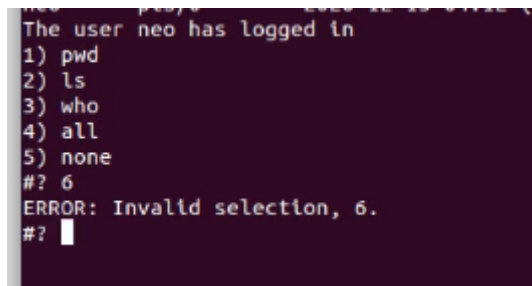
şeklinde kullanılır.

-her bir item bir numara ile gösterilir. seçim yapılması beklenir

-seçim için bir giriş verilmesi gerekir.

----- örnek

```
select op in pwd ls who all none    // bunlardan birini seç diyor.
do
    case $op in    // değerleri okuyor
        pwd| ls |who ) eval $op ;;    // değeri okuyup çalıştırıyor komutu
        all) pwd; ls; who;; // all ise hepsini koşuyor
        none) break ;; // none ise çıkış yapar
        *) echo "ERROR: Invalid selection, $REPLY." ;;    // başka bir şey ise sistemde tanımlanan değişkenden değer döner
    esac
done
```



çıktı bu şekilde olur.

\$REPLAY → * olan duruma girilen değeri döner. sistem tarafından tanımlanmış bir değişkendir.

export -> çevre değişkenlerine ekleme yapar.

basename: girilen pathın son değişkenini döndürür. en sondaki dosya adı gelir.

parametreleri vardır. Ama hoca değinmemiş.

\$> basename /usr/bin/sh değer olarak sh döner.

----- printf

echodan farklı olarak \n bastırıyor.

echoda \n var printf de yok.

printf ecohoya göre biçimsel değişkenleri kullanabilir. %i, %d, %s gibi gibi

echo bunları bastırmaz. yapamıyor daha küçük :D büyüyünce echoda yapabilir belki.

printf format arguments

format: %[-]m.nx -> %32s, %7f

x -> tipi için kullanılır. s, c, f, i, d vs vs

m -> min, **n** -> max değerdir.

s	String
c	Character
d	Decimal (integer) number
x	Hexadecimal number
o	Octal number
e	Exponential floating-point number
f	Fixed floating-point number
g	Compact floating-point number

9. hafta 2. ders

----- örnek

```
#!/bin/bash
printf "%32s %s\n" "File Name" "File Type" // 32s ile file name %s ile file type eşleşir.
for i in *; // bulunduğumuz dizindeki dosyaları sırayla alır.
do
    printf "%32s " "$i" // dosya adını yazar
    if [ -d "$i" ]; then echo "directory" // düzenli dosya ise directory yazar.
    elif [ -h "$i" ]; then echo "symbolic link" // sembolik linkse -l de aynı işlemi yapar.
    elif [ -f "$i" ]; then echo "file" // file ise
    else echo "unknown" // bilinmiyorsa
    fi; // if biter.
done // döngü biter.
```

%32s -> 32 birimlik boşluktan sağa doğru yazılır veri. 32 karakterse soldan başlamış gibi görünür.

Celal5 dosyası

```
#!/bin/bash
printf "%52s %s\n" "File Name" "File Type"
for i in *;
do
    printf "%52s " "$i" // print ilr dosya adı yazdırılır.
    if [ -d "$i" ]; then echo "directory" // echo ile dosya tipi yazılır ve alt satıra iner.
    elif [ -L "$i" ]; then echo "symbolic link"
    elif [ -f "$i" ]; then printf $i | wc -c
    else echo "unknown"
    fi
done
```

```
neogubuntu:~$ bash celal5
File Name File Type
awkkodu- 8
celal2 6
celal3 6
celal4 6
celal4- 7
celal5 6
cikktici 7
Desktop directory
Documents directory
dosya- 6
dosya1- 7
Downloads directory
Music directory
Pictures directory
Public directory
reguler- 8
reguler-c- 10
soru2.sh- 9
Templates directory
Videos directory
neogubuntu:~$ ^C
neogubuntu:~$
```

çıktı.

---- ödevler:

dfs

wc

argumanları yazan for döngüsü

celal5

10. hafta 1. ders;

----- tee komutu

çıktı bir dosyaya, ekrana ya da pipe a aktarılabilir.

işlemi > gibidir.

date | tee now -> data içeriği now içine aktarılır.

ls | tee list | wc -> ls çıktısının wc içeriği elde edilir.

ps -ael | tee processes | grep "\$UID" -> çalışan prosesler prosese aktarılır. \$UID aranır.

herhangi bir değişkenin değeri de aranabilir.

-a -> başkalarına ait olan prosesleri gösterir

-e = -A -> bütün prosesler seçilir.

-l -> prosesleri listeler.

Dosyaları dosya tanımlayıcılarıyla ilişkilendirme **Associating Files with a File Descriptor**

standart giriş (STDIN), 0 dışardan giriş alınır.

standart çıkış (STDOUT), 1 terminale çıktı verir.

standart Error (STDERR), 2 terminale çıktı verir.

her komut için 3 tane dosya işaretçisi vardır. Sistem tarafından tanımlıdır.

bu dosyalarla ilişkilidir ama isteğe bağlı olarak bir dosyaya aktarılabilir.

yazma modunda bir dosya açmak için;

- `exec n>file`
- `exec n>>file`

okuma modunda açmak için;

- `exec n<file`
- `exec n<<file`

hem yazma hem de okuma modunda açmak için;

- `exec n<>file`
- `exec n<<>>file`

Çalışma şekli

1. dosya varsa dosyanı içeriğini temizler.
2. o dosya ekleme modunda açılır.

n değerleri;

n bir integer değerdir.

n = 0 ise giriş | okuma işlemi yapar. verilen dosyanın içeriğini okur.

n=1 ise çıktı | standart çıktı verilen dosyaya yönlendirilir.

n=2 ise error

exec 5>zor // dosya yoksa oluşturur. varsa içeriğini temizler.

ps aux 1>&5 // normal çıktıyı yönlendirir. 5 pointerına yani zor dosyasına. &5 yerine zor yazılabilir.

date 1>>zor // date içeriği zora eklenir. 1 olmasa da olur.

exec 6<file // file ile 6 ilişkilendirilmiştir. okuma modunda

exec 7<zor // zor okuma modunda 7 ile ilişkilendirilir.

cat 0<&6 // 6 ile ilişkilendirilen dosyayı cat ile oku

cat 0<&7 // 7 yi oku

if [-f zor] // zor dosya olarak var mı

i=0 // var i= 0

while read LINE // satırı oku

do

i = `echo "\$i + 1" | bc` // i bir artar. ekrana yazar. pipe ile bc komutuna gönderilir.

done<zor // bu dosya üzerinden okuma işlemi yapılacak. read ile okunup LINE a aktarılacak

echo \$i // i değeri döner yani satır sayısı bulunur.

fi

bc = exact içine aktarılan komutta matematik işlemi varsa onu yapar.

```
dizincarcala x path x centos x processes x dosya-satir-sayisi x file x
1 #!/bin/bash
2 exec 5>zor
3 ps aux 1>&5
4 date 1>>zor # date >> zor
5 exec 6<file
6 exec 7<zor
7 cat 0<&6 # cat <&6
8 cat 0<&7 # cat <&7
9 if [ -f zor ]
10 then
11     i=0
12     while read LINE
13     do
14         i=`echo "$i + 1" | bc`
15     done < zor
16     echo $i
17 fi
18
```

```
neo@ubuntu:~$ echo selan
selan
neo@ubuntu:~$ echo selan > processes
neo@ubuntu:~$ clear
neo@ubuntu:~$ bash dosya-satir-sayisi
deneme-
deneme2-
deneme3-
deneme3.cpp-
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1   0.0  0.1 33892 4324 ?        Ss   Dec20   0:01 /sbin/init
root         2   0.0  0.0   0     0 ?        S    Dec20   0:00 [kthreadd]
root         3   0.0  0.0   0     0 ?        S    Dec20   0:00 [ksoftirqd/0]
root         5   0.0  0.0   0     0 ?        S<   Dec20   0:00 [kworker/0:0H]
root         7   0.0  0.0   0     0 ?        S    Dec20   0:43 [rcu_sched]
root         8   0.0  0.0   0     0 ?        S    Dec20   0:00 [rcu_bh]
root         9   0.0  0.0   0     0 ?        S    Dec20   0:00 [migration/0]
root        10   0.0  0.0   0     0 ?        S    Dec20   0:01 [watchdog/0]
root        11   0.0  0.0   0     0 ?        S    Dec20   0:01 [watchdog/1]
root        12   0.0  0.0   0     0 ?        S    Dec20   0:00 [migration/1]
root        13   0.0  0.0   0     0 ?        S    Dec20   0:00 [ksoftirqd/1]
root        15   0.0  0.0   0     0 ?        S<   Dec20   0:00 [kworker/1:0H]
root        16   0.0  0.0   0     0 ?        S    Dec20   0:00 [kdevtmpfs]
root        17   0.0  0.0   0     0 ?        S<   Dec20   0:00 [netns]
```

zor dosyasının içeriği

```
root      58623   0.0  0.0   0     0 ?        S    17:28   0:00 [kworker/0:0]
neo       58649   0.0  0.0 16624 2744 pts/0    S+   17:32   0:00 bash dosya-sat
ir-sayisi
neo       58650   0.0  0.0 22644 2676 pts/0    R+   17:32   0:00 ps aux
Tue Dec 22 17:32:59 MSK 2020
263
neo@ubuntu:~$
```

toplam satır sayısını gösterir.

for FILE in \$FILES

do

ln -s \$FILE ./docs >> /tmp/ln.log 2> /dev/null // s. link eklenir. hata verirse dev null'a gider.

done

10. hafta 2. ders

ln -s -> hedef directory, sadece hedef,

ls -s -> dosya s.link adı

hedef olarak

path belirtilecekse -s kaldırılır. aynı isimle s.link eklenir.

command > *file* 2>&1 -> hata ve çıktı 1 ile ilişkilendirilen dosyaya aktarılır.

----- Standart error çıktısı

echo *string* 1>&2 -> çıktı terminale gelir.

printf *format args* 1>&2 -> çıktı terminale gelir.

if [! -f \$FILE] ;

then echo "ERROR: \$FILE is not a file" >&2 ;

fi -> dosya değilse hata retür. terminale yazar.

----- Fonksiyonlar

fonksiyon_adi () {} şeklinde tanımlanır.

source mycd-> ile fonksiyon dosyası terminale tanıtılır. bağlantı kurulur.

----- örnek

lspath() { // fonk adı

OLDIFS="\$IFS" // değişkene çevre değişkeninin değeri verilir. değeri boşluktur.

IFS=: // dizinler arasında : kullanılıyor. normalde " " dur ayırma karakteri

for DIR in \$PATH ; // path çevre değişkeninden parçalama yapar.

do

echo \$DIR ; // ayrılan dizin ekrana yazdırılır. path değeri ayrılır. tek tek

done

IFS="\$OLDIFS" // sistemin değişkeni orijinal halini kaybetmesin diye eski değeri geri verilir.

}

```

1 lspath()
2 {
3     OLDIFS=$IFS
4     IFS=:
5     for DIR in $PATH ; do echo $DIR ; done
6     IFS=$OLDIFS
7 }
8
9 neo@ubuntu:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games
10 neo@ubuntu:~$ source dizinparcala
11 neo@ubuntu:~$ lspath
12 /usr/local/sbin
13 /usr/local/bin
14 /usr/sbin
15 /usr/bin
16 /sbin
17 /bin
18 /usr/games
19 /usr/local/games
neo@ubuntu:~$

```

sonuç

lspath | grep "/usr/bin/" -> fonksiyonu koş, verilen dizini ara. önce source ile fonk terminale tanıtılmalı.

----- örnek

```

SetPath() {
    PATH=${PATH:="/sbin:/bin"}; // Path değişkenine verilen değeri atar.
    for _DIR in "$@" // dir değişkeni gelen parametreleri dizi şeklinde alır.
    do
        if [ -d "$_DIR" ]; // dir dizinde
        then PATH="$PATH": "$_DIR" ; // girilen dizini path'e aktarır. aralarına : ekler
        fi
    done
    export PATH // yeni path değeri export edilir. geçerli terminal boyunca kullanılır.
    unset _DIR // dir yok edilmiş.
}

```

```

neo@ubuntu:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games
neo@ubuntu:~$ source dizinparcala
neo@ubuntu:~$ lspath
/usr/local/sbin
/usr/local/bin
/usr/sbin
/usr/bin
/sbin
/bin
/usr/games
/usr/local/games
neo@ubuntu:~$ SetPath /dev/block /etc/acpi /media/neo
neo@ubuntu:~$ lspath
/usr/local/sbin
/usr/local/bin
/usr/sbin
/usr/bin
/sbin
/bin
/usr/games
/usr/local/games
/dev/block
/etc/acpi
/media/neo

```

iki fonk ile path e yeni değişkenler eklenmiş oldu

11. hafta 1. ders

---- fonksiyonlara parametre verilmesi

- **function-name** arg1 arg2 arg3 argN shellde olduğu gibi argüman verilir.

- \$ vi [pass](#)

```
function demo()  
{  
    echo "All Arguments to function demo(): $*"   
    echo "First argument $1"    // 1. argüman  
    echo "Second argument $2" // 2. argüman  
    echo "Third argument $3"   // 3. argüman  
    return  
}  
# Call the function  
demo -f foo bar // çağırma işlemi
```

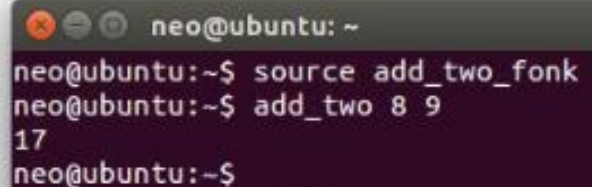
----- örnek

```
function add_two { // iki değer alır ve onu toplar  
  
    (( sum=$1+$2 ))  
  
    return $sum // sonucu döner  
  
}
```

add_two 1 3 // çağırma işlemi

echo \$? // dönen değeri dönüştürür. hata varsa -1 ya da 0, hata yoksa 1 ya da daha yüksek değer döner.

```
1 function add_two()  
2 {  
3     (( sum= $1 + $2 ))  
4     echo $sum  
5 }  
6 #  
7 # sonuc=$(add_two $1 $2)  
8 # echo $sonuc
```

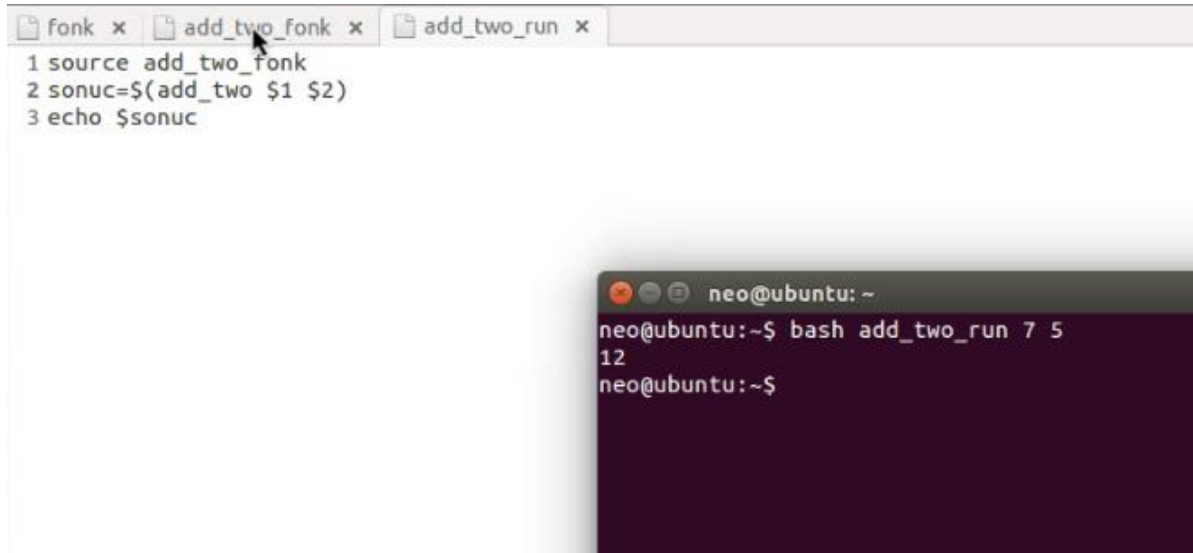


```
neo@ubuntu: ~  
neo@ubuntu:~$ source add_two_fonk  
neo@ubuntu:~$ add_two 8 9  
17  
neo@ubuntu:~$
```

fonksiyonun çağırılması.

- fonksiyonun kullanılmadan önce tanımlanması gerekiyor.

Script üzerinden fonksiyon çağırılması;



```
funk x add_two_funk x add_two_run x
1 source add_two_fonk
2 sonuc=$(add_two $1 $2)
3 echo $sonuc

neo@ubuntu: ~
neo@ubuntu:~$ bash add_two_run 7 5
12
neo@ubuntu:~$
```

parametre verilmezse hata verir.

- Scripte çalıştırılırken fonksiyondan dönen değer bir değişkene aktarılır. değişken yazdırılır.
- Fonksiyonu tanıtırken argüman verilmezse; hata verir ve run time errordür. (.exe üretilir ama koşarken hata verir.) Tekrar tanıtmadan argümanla çalıştırılırsa hata vermeden çalışır.
- **echo \$? //** dönen değeri dönüştürür. hata varsa -1 ya da 0, hata yoksa 1 ya da daha yüksek değer döner.

----- Yığın işlemleri

popd, pushd, dirs

- **pop** = ekle
- **push** = çek
- **dirs** = tümünü listele

-----Pushd > boş yığına veri ekleme

pushd() { // bir argüman lacak dizinse yığına ekleyecek

REQ="\$1"; // girilen değeri aldık rq ya ttık

if [-z "\$REQ"] ; // boş mu

then REQ=. ; // boşsa bulunduğun yerde kal

fi

if [-d "\$REQ"] ; then // parametre dizinmi

cd "\$REQ" > /dev/null 2>&1 // dizinse oraya git hata ya da çıktı varsa ekrana çıktı verme

if [\$? -eq 0] ; then // son koşan komut (\$?) 1 mi 0 mı (0 true, 1 false) cd değişmişse yani

_DIR_STACK="`pwd`:\$ _DIR_STACK" ;

// işlem başarılıysa yığına ekle bulunduğun dizini göster dizinin değişmiş hali

export _DIR_STACK ; dirs

// _DIR_STACK bunu öncekinin üstüne ekle yığın mantığı yeni gelen üste eklenir. sonra dirs fonk çalışır

else

echo "ERROR: Cannot change to directory \$REQ." >&2

// cd çalışmadıysa hata verecekyetki nedeniyle girilmemiş olabilir.

fi

else

echo "ERROR: \$REQ is not a directory." >&2 // girilen argüman dizin değil

fi

unset REQ // req ile işi bitti ve değişkeni yok etti

}

-----Dirs

```
dirs() {  
    OLDIFS="$IFS" // çevre değişkeni olan IFS yi oldifs de tutuyoruz. daha sonra düzeltereğiz  
    IFS=: // boşluk olarak belirli olan ifs ye : değeri verilir.  
    for i in $_DIR_STACK // daha önce push ile eklediğimiz yığın elemanları for a aktarılır.  
    do  
        echo "$i \c" // yığınlar son eklenenden başlanıp sıralanır. \c -> kursor aynı satırda kalır.  
    done  
    echo // alt satıra inme olmazsa kullanıcı adımız aynı satırın sonuna eklenir.  
    IFS="$OLDIFS" // IFS eski haline geri döner.  
}
```

Sonuç olarak \c den dolayı yan yana yazılır değerler.

\c -> kursor aynı satırda kalır

echoya -e parametre de verilmeli. bu sayede \x durumunda x karakteri yorumlanır.

```
neo@ubuntu:~$ source fonk  
neo@ubuntu:~$ dirs  
  
neo@ubuntu:~$ pushd  
/home/neo  
neo@ubuntu:~$ dirs  
/home/neo  
neo@ubuntu:~$
```

ilk ekleme

```
neo@ubuntu:~$ source fonk  
neo@ubuntu:~$ dirs  
  
neo@ubuntu:~$ pushd  
/home/neo  
neo@ubuntu:~$ dirs  
/home/neo  
neo@ubuntu:~$ pushd /  
/ /home/neo  
neo@ubuntu:/$ pushd /dev  
/dev / /home/neo  
neo@ubuntu:/dev$ pushd /etc  
/etc /dev / /home/neo  
neo@ubuntu:/etc$ pushd /usr/bin  
ERROR: /usr/bin is not a directory.  
neo@ubuntu:/etc$ pushd /usr/bin  
/usr/bin /etc /dev / /home/neo  
neo@ubuntu:/usr/bin$ dirs  
/usr/bin /etc /dev / /home/neo  
neo@ubuntu:/usr/bin$
```

ekleme yapıldıkça eskisinin önüne yazılır.

-----Popd

```
popd() {  
    OLDIFS="$IFS" // ifs tekrar değiştirilir.  
    IFS=:  
    _popd_helper $_DIR_STACK // yığından işlem yaparak eleman alını. popdhelper da bir fonk.  
    IFS="$OLDIFS"  
}  
  
_popd_helper() {  
    POPD="$1" // üstteki fonksiyondan _popd_helper $_DIR_STACK değeri alınır. yığının tamamı  
    if [ -z "$POPD" ] ; then  
        echo "ERROR: The directory stack is empty." >&2 // ilk eleman boşsa ekrana çıktı  
        veriyor.  
        return 1  
    fi  
    shift // 2. elemana geçiliyor.  
    if [ -n "$1" ] ; then // yığın boş değil mi? yığın tek elemanlı mı anlamını da taşıyor.  
        _DIR_STACK="$1" ; // boş değil 2. eleman tutuluyor.  
        shift ; // tekrar kaydırma işlemi yapıyor. 3. elemana geçtik.  
        for i in $@ ; // geriye kalan elemanları yazdırır.  
        do _DIR_STACK="$_DIR_STACK:$i" ;  
            // 2. eleman kaybolmasın diye tuttuk tekrar yığına ekledik  
        done  
    else // yığında 1 eleman vardı $1 in değeri boşsa burası çalışır.  
        _DIR_STACK=  
    fi  
    if [ -d "$POPD" ] ; then // popd bir dizinse  
        cd "$POPD" > /dev/null 2>&1 // tekrar dizin değiştirilir. (eklenen her şey dizindi)  
        if [ $? -ne 0 ] ; then
```

// dirste bu durum kontrol edilip eklenmişti izin değiştirilmişse ya da silinmişse bu durum geçerli olur.

echo "ERROR: Could not cd to \$POPD." >&2 // dizini değiştiremedi

fi

pwd // işlem başarılıysa bulunduğu yer

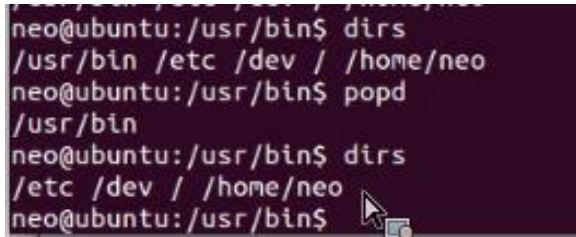
else

echo "ERROR: \$POPD is not a directory." >&2 //çekilmek istenen değer dizin değilse buraya gelir

fi

export _DIR_STACK // export edilerek

unset POPD // yok edilir.



```
neo@ubuntu:/usr/bin$ dirs
/usr/bin /etc /dev / /home/neo
neo@ubuntu:/usr/bin$ popd
/usr/bin
neo@ubuntu:/usr/bin$ dirs
/etc /dev / /home/neo
neo@ubuntu:/usr/bin$
```

- yığın ilk satırda **dirs** çalıştırılmasıyla ekrana basıldı.
- **popd** ile ilk eleman çekildi.
- son satırda yığının son hali var.

NOTTTTTTTTTT; shift kullanıldığında 1. eleman 2. eleman olur; 2. eleman 3. eleman olur. kaydırma işlemi olduğu için.

Kodda \$1 ilk önce tüm yığını tutuyordu, daha sonra shift satırında elemanlar kaydı ve \$1 yığının 2. elemanını tutmaya başladı. (neden 2. eleman kaydırma işlemi yaptığı için.)

11. hafta 2. ders

kodda **POPD="\$1"** ile ilk eleman tutuluyor.

echo [options] [string, variables...]

-n -> bulunduğu satırda kalmamızı sağlar **echo -n = printf**

-e -> daha sonra eklenen\ ifadesi okunur.

- \c suppress trailing new line
- \a alert (bell) - **bip sesi verir -n ile kullanılabilir.**
- \b backspace -
 \n new line - **echo -e \n = echo -n = printf**
- \r carriage return -
 \t horizontal tab – **elemanlar arası yatayda tab kadar boşluk bırakır.**
- \\ backslash – **bir tane \ yazmış oluruz.**

-----Reklendirmee

echo -e "\033[34m Hello Colorful World!" // 033 escape karakter.

[34m mavi yazar

30 – 40 ' a kadar ve daha fazla renk var. bu şekilde renkli yazar.

0m bol yazar.

1m

2m veri eski haline döner

7m arkaplan beyaz yazı rengi siyah

5m çalışmıyormuş

h	ANSI modunu ayarlayın
l	ANSI modunu temizler
m	Karakterleri farklı renklerde gösterin veya BOLD ve Blink gibi efektler
q	Turns keyboard num lock, caps lock, scroll lock LED on or off
s	Stores the current cursor x, y position (col , row position) and attributes
u	Restores cursor position and attributes

h l m q Klavye num kilidini açar, büyük harflerle kilit, kaydırma kilidi LED'i açık veya kapalı s Mevcut imleç x, y konumunu kaydeder (sütun, satır konumu) ve öznitelikler u İmleç konumunu ve niteliklerini geri yükler

----- Sistem programlama

gcc, g++ -> derleyiciler

g++/gcc hello.cpp

yapılırsa a.out'a çıktı iletilir. Bunun yapılması tavsiye edilmeeez

g++ -o hello hello.cpp

o parametresi ile hello.cpp dosyası koşulur. hello adlı dosyadan çıktıya bakılır.

g++ -o hello hello.cpp util.cpp

hello.cpp ve util.cpp koşulur ve çıktı hello ya aktarılır. ikisi beraber derlenir

g++ -c hello.cpp

g++ -c util.cpp

g++ -o hello hello.o util.o // -o hem derleme hem de dosya üretme

g++ -c compile edilir. sadece derler. objeyi oluşturur.

g++ -o <file> çıktı eklenen dosyaya aktarılır. genelde aynı dosya ismi verilir. verilmezse a.out oluşur. onun üstüne yazılır.

g++ -D makro üretir

g++ -I kütüphaneler eklenir.

g++ -I (i) verilen dizin üzerindeki dosyaları ekler

g++ -L verilen dizin üzerindeki kütüphaneler kullanılabilir.

#ifdef DEBUG

printf("value of var is %d", var); // debug modundaysa bu satırı çalıştır. değilse iften çık

#endif

g++ -DDEBUG -o prog prog.c // debug modunda çalıştırma

---- **make**

bütün dosyaların düzgünce çalışması.

dosyaların izlenmesi (değiştirilmesi)

dosyalar arası bağımlılıklar. bunları otomatik yapar

[Mm]akefile şeklinde çalışır.

bağılılık zincirinde en bağımsızdan en bağımlıya doğru hareket edilir.

kendisine bağlı olan alt dosyalar oluşturulur

Makefile for mydb

```
mydb: mydb.o user.o database.o //makefile
```

```
g++ -o mydb mydb.o user.o database.o // dosyalar yazılıyor.
```

```
mydb.o : mydb.cpp mydb.h // bağıladı
```

```
g++ -c mydb.cpp // taba basılmış sonra komut
```

```
user.o : user.cpp mydb.h // bağıladı bağımlılık satırı. target : bağımlılık
```

```
g++ -c user.cpp
```

```
database.o : database.cpp mydb.h // bağıladı
```

```
g++ -c database.cpp
```

bağımlılık satırı:

tabla başlar

\$@ target hedefi listeler

\$? dependencies bağımlılıkları listeler

----- macros çok değinmemiş -D ile oluşturuluyordu.

tekrar bakılması gereken bir konudur.

:P bittiiiiiiiiimm

SED

*Stream editörün kısaltmasıdır.

*Ed line editörden türetilmiştir.

*Sed '3,4p' foo.txt // İlgili verilen akış üzerinde verdiğimiz aralığın duplicate(tekrardan yazılması) etmeyi sağlıyor. (tek parametre ile çalışmıyor)

*sed '4q' foo.txt dosyanın başından itibaren verilen satıra kadar yazdırma işlemi yapar ve geri kalanı yazdırılmaz. Mesela 4. Satırda basmayı bırakacak bu kod.

* sed -n '3,4p' foo.txt sadece verilen aralığın gösterilmesini sağlar. Sadece ilgili kısmı getiriyor.

*sed -n '\$p' foo.txt ilgili dosyanın sadece son satırını getirdi.

*sed -n '3,\$!p' foo.txt verilen aralıktaki satırların dışındaki satırları getirir. 3. Satırdan sonrakileri getirir.

NOT: kısacası -n verirse bastırıyor, -n yazmazsak duplicate yapıyor.

SED DÜZENLİ İFADELER İLE KULLANIMI

*sed -n '/^ FROM: /p' \$HOME/mbox "\$HOME/mbox" dosyası üzerinde " From:" (<BOŞLUK> From:) ile başlayan satırları bana göster diyoruz.

Ls -l | sed -n '/^.....w/p' Başlangıçtan itibaren 5 karakter ne olursa olabilir 6. karakter "w" olsun. Ls çıktısında 1. Değer dosya tipi, diğerleri 3er ugo. Yani aslında grubun yazma hakkın sahip dosyalarını listelemiş olduk.

Sed:Substition DEĞİŞTİRME İŞLEMİ

Sed 's/|/:/g' data.txt burada slashlar arasında kalan karakterleri değiştirir. Yani her bulduğu "/" için ":" koyar. Buradaki "g" aramanın sürekli yapılacağı anlamına geliyor. Yani işlemi yaptıktan sonra bir sonraki işlem kaldığımız yerden devam eder. "g" yi vermezsek en baştan tekrar arar

AWK

Aho, Weinberger ve Kerninga oluşturan kişiler ve baş harflerinden oluşuyor.

Awk -f awk.script foo.txt

Begin içerisine, belirtilen dosyayı okuma işlemine geçmeden önce yapmak istediğimiz işlemleri buraya yazıyoruz.

Ortadaki süslü parantezler içerisinde ise, alınan dosyanın HER BİR SATIRI için gerçekleştirilecek işlemleri yapıyoruz.

End içerisinde ise Okuma işlemi bittikten sonra gerçekleştirilecek işlemlerin belirtildiği yerdir.

Örnek:

```
# Begin Processing
BEGIN {print "Print Totals"}

# Body Processing
{total = $1 + $2 + $3}
{print $1 " " + " $2 " + " $3 " = "total"}

# End Processing
END {print "End Totals"}
```

Yukarıdaki örnekte dolarlar ile belirtilen yerler sütunlara tekabül eder.

awk_satislar	awk_sayilar
1 1 clothing 3141	1 22 78 44
2 1 computers 9161	2 66 31 70
3 1 textbooks 21312	3 52 30 44
4 2 clothing 3252	4 88 31 66
5 2 computers 12321	
6 2 supplies 2242	
7 2 textbooks 15462	

Yukarıdaki kod 1. 2. Ve 3. Sütunları toplayarak total e atıyor ve onu print ediyor.

```
1 BEGIN {print "computers"}
2 {
3   if ($2 == "computers")
4     print $3
5 }
6 # End Processing
7 END {print "son"}
```

yandaki kod parçasında 2. Sütundaki her eleman computer mi diye kontrol ediyor, eğer öyle ise gidip fiyatını yazdırıyor. Tüm computerler bitince de son yazdırıyor.

Yukarıdaki kodu koşturmak için;

Awk -f awk_computers awk_satislar yazıyoruz. Çıktı:

```
neo@ubuntu:~$ awk -f awk_computers awk_satislar
computers
9161
12321
son
neo@ubuntu:~$
```

AWK Döngüleri

```
awk_satislar x  awk_sayilar x  awk_satislari_toplama x
1 BEGIN {OFS = "\t"}
2 {deptSales [$2] += $3}
3 END {for (item in deptSales)
4     {
5         print item, ":", deptSales[item]
6         totalSales += deptSales[item]
7     } # for
8     print "Total Sales", ":", totalSales
9 } # END
```

Yukarıdaki kod parçasında OFS bir çevre değişkenidir ve bu değerin default değeri tek bir boşluktur, eğer biz bunu değiştirsek yazdırma anında elemanlar arasındaki karakteri değiştirmiş oluyoruz. Yani kısaca basma işleminde kullanılacak işlem boşluk değil, "TAB" olacağı için (\ t) tab bırakacak aralara. Sütunsal bazda hizalama yapmak için.

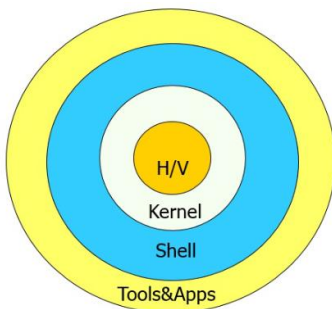
Her bir turda yani ilk satırda icra edilecek kısım deptSales[\$2], gidip clothingı alıyor sonra değerini topluyor, 2. Adımda gidip computersi topluyor, yani aslında deptSales[computers] gibi bir indisin içerisine değer atıyor. Böyle bir durumda Computers için listedeki tüm computersleri toplayarak tek bir index içinde toplam değerlerini toplamış oluyor.

For döngüsü içerisinde itemleri alıyor, aldığı itemleri yazdırıyor, sonrasında itemin bulunduğu indexi diziye verip o birimin toplam satışını görmüş oluyoruz. Total sales içerisinde ise bütün birimlerin kendi değerlerini tekrar toplayarak totaldeki satışını oluşturuyor, for bitince de toplam yapılan bütün genel satış değerini yazdırıyor.

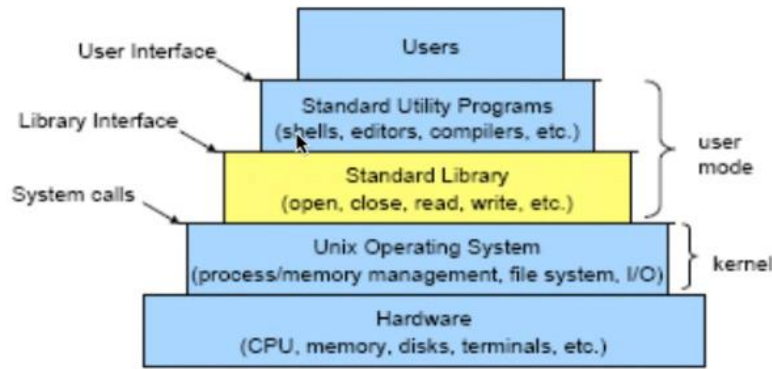
Çıktı:

```
neo@ubuntu: ~
neo@ubuntu:~$ awk -f awk_satislari_toplama awk_satislar
supplies      :      2242
computers     :     21482
textbooks     :     36774
clothing      :      6393
Total Sales   :     66891
neo@ubuntu:~$
```

FILE MANAGEMENT



Unix tabanlı sistemlerin katmanları:



UNIX SİSTEM PROGRAMLAMA

Programlar sistem çağrılarını kullanarak kütüphanelere erişirler.

Sistem çağrı tipleri:

- *File I/O dosya giriş çıkışları
- *Process Management process yönetimi
- *Inter Process Communication(IPC)
- *Signal Handling

C dili vs C++

C de string tipi diye bişey yok.

C için;

- *Strcpy() kopyalıyor
- *Strcmp() karşılaştırma yapıyor.

C++ de printf ve cout var

C de scanf ve fgets() var

BASIT DOSYA GİRİŞ ÇIKIŞLARI I/O

Dosyalar okuma veya yazma modunda açılabilir.

Bu sistem çağrılarını kullanabilmek için <stdio.h> (Standard input output kısaltması) include etmemiz lazım

Her bir dosya bir dosya işaretçisi tarafından referanslanır.

3 tane dosya otomatik olarak açılır:

FD 0 standart input

FD1 standart output

FD2 standart error

Open() FONKSİYONU

- `fd = open(path, flags, mode);`
- `path`: `char*`, absolute or relative path
- `flags`:
 - `O_RDONLY` – open for reading
 - `O_WRONLY` – open for writing
 - `O_RDWR` – open for reading and writing
 - `O_CREAT` – create the file if it doesn't exist
 - `O_TRUNC` – truncate the file if it exists (overwrite)
 - `O_APPEND` – only write at the end of the file
- `mode`: specify permissions if using `O_CREAT`
- Returns newly assigned file descriptor
- `fd = open("myfile", O_CREAT, 00644)`

Open içindeki flag dediği şey dosya tipidir. Mode hangi modda açılacağı. Path ise dosyanın yolunu verir.

`O_RDONLY` yalnızca okunur

`O_WRONLY` sadece yazılabilir.

`O_RDWR` okunabilir ve yazılabilir.

`O_CREAT` öyle bir dosya yoksa oluşturur

`O_TRUNC` dosya varsa içeriğini boşaltır

`O_APPEND` dosyanın sonuna ekler.

`Fd= open("myfile", O_CREAT, 00644)` burada program koşunca myfile diye dosya oluşturacak sürekli.

Read() FONKSİYONU

- `bytes = read(fd, buffer, count);`
 - Read from file associated with `fd`; place `count` bytes into `buffer`
 - `fd`: file descriptor to read from
 - `buffer`: pointer to array of `count` bytes
 - `count`: maximum number of bytes to read
 - Returns the number of bytes read or -1 on error
- ```
int fd = open("someFile", O_RDONLY);
char buffer[4];
int bytes =
 read(fd, buffer, 4);
```

- 1.parametresi Okuma yapacağımız dosya tanımlayıcısı.
- 2.parametresi Okunacak verilerin aktarılacağı dizidir.(buffer)
- 3.parametresi Dosyadan kaç tane veri okuyacağımızı gireriz. (genelde buffer kadar beklenir ama daha az da olabilir.)

---

## Write() FONKSİYONU

- `bytes = write(fd,buffer, count)`
  - Write contents of `buffer` to a file associated with `fd`
  - `fd`: file descriptor
  - `buffer`: pointer to an array of `count` bytes
  - `count`: number of bytes to write
  - Returns the number of bytes written or `-1` on error
- ```
int fd = open("someFile", O_WRONLY);
char buffer[4];
int bytes =
    write(fd, buffer, 4);
```

`Close()` fonksiyonu ile işleminiz bitince kapatabilirsiniz dosyayı.

`Lseek()` fonksiyonu ile arama işlemleri yapılır.

`lseek()`

- `retval = lseek(fd,offset,whence)`
- Move file pointer to new location
- `fd`: file descriptor
- `offset`: number of bytes
- `whence`: .
 - `SEEK_SET` – offset from beginning of file
 - `SEEK_CUR` – offset from current offset location
 - `SEEK_END` – offset from end of file
- Returns offset from beginning of file or `-1` on error

2. parametre Offset değeri dosya içinde nerede olduğumuz yer.

3. parametre offset değerinin baz alınacağı durum.

`SEEK_SET` Dosyanın başını gösteriyor

`SEEK_CUR` aradığımız veriyi bulunca kaldığımız yerden devam edebilmemiz için o konumda `CUR` kullanılıyor.(recursive bi mantıkla dosya sonuna kadar gidebilmek için)

`SEEK_END` aynı şekilde `CUR` gibi ancak sondan arama yapıyor.

fopen ()

- `FILE *file_stream = fopen(path, mode);`
- `path`: `char*`, absolute or relative path
- `mode`:
 - `r` – open file for reading
 - `r+` – open file for reading and writing
 - `w` – overwrite file or create file for writing
 - `w+` – open for reading and writing; overwrites file
 - `a` – open file for appending (writing at end of file)
 - `a+` – open file for appending and reading
- `fclose(file_stream);`
 - Closes open file stream

*r dosyayı okuma modu

*r+ okuma yazma modu

*w dosyayı yazma modunda açıyoruz, dosya yoksa oluşturur

*w+ modunda açarsak dosyayı hem okuma hem yazma modunda açıyoruz, eğer dosya varsa da içeri boşaltıyoruz.

*a dosyayı sondan eklemek için açıyor

*a+ dosyayı hem sondan eklemek hemde okumak amacıyla açıyor.

Fclose ile içine verilen dosyayı kapatıyor kod içinde.

*Fprintf formatlı yazmamıza yarıyor

```
1 /* fprintf example */
2 #include <stdio.h>
3
4 int main ()
5 {
6     FILE * pFile;
7     int n;
8     char name [100];
9
10    pFile = fopen ("myfile.txt","w");
11    for (n=0 ; n<3 ; n++)
12    {
13        puts ("please, enter a name: ");
14        gets (name);
15        fprintf (pFile, "Name %d [%-10.10s]\n",n+1,name);
16    }
17    fclose (pFile);
18
19    return 0;
20 }
```

dümdüz yazmak yerine araya boşluklar koy, tab koy

vs onun için kullanılıyor. 3 parametre alır, 1. Parametre nereye yazılacak, 2. Parametre formatlı yazım tipi, sonra da yazdırılacak elemanlar.

*Fputs 2 parametre alıyor, 1. Parametre yazdırılacak pointer dizisi, 2. Parametre ise yazdırılacak dosya.

```
1 /* fputs example */
2 #include <stdio.h>
3
4 int main ()
5 {
6     FILE * pFile;
7     char sentence [256];
8
9     printf ("Enter sentence to append: ");
10    fgets (sentence,256,stdin);
11    pFile = fopen ("mylog.txt","a");
12    fputs (sentence,pFile);
13    fclose (pFile);
14    return 0;
15 }
```

FILE tipinde bir dosya değişken oluşturulmuş. 256 karakterlik bir dizi oluşturulmuş. Eklemek istediğiniz diziyi ekleyin diyor, standart girişten (stdin) alıyor ve 256 birimlik olarak. Fgets 3 parametre alıyor. 1. Parametre dizi gelmesi lazım pointer veya normal dizi, 2. Parametrede dosyadan çekilecek karakter sayısını belirtiyoruz.

konsoldan yazılanın 256 karakterini alıyor, aldıktan sonrada sentence isimli diziye aktarıyor fgets ile. Eğer 256 karakteri aşmazsam konsoldan girilen ifade diziye yerleşmiş oluyor. File tipinde bir pointer tanımlanmıştı, fopen ile onu açtı

*Fopen 2 parametre alıyor, 1. Si dosya adı, 2.si ise hangi modda açılacağı. var olan dosya varsa içini boşaltır, yoksa içini

*Fputc ve fgetc karakter okuma ve koyma yapıyorlar.

*fopen dosya varsa açar yoksa oluşturup açar

* while(c!=EOF) dosya sonuna gelene kadar demek.

```

1 /* fscanf example */
2 #include <stdio.h>
3
4 int main ()
5 {
6     char str [80];
7     float f;
8     FILE * pFile;
9
10    pFile = fopen ("myfile.txt","w+");
11    fprintf (pFile, "%f %s", 3.1416, "PI");
12    rewind (pFile);
13    fscanf (pFile, "%f", &f);
14    fscanf (pFile, "%s", str);
15    fclose (pFile);
16    printf ("I have read: %f and %s \n",f,str);
17    return 0;
18 }

```

Rewind pozisyon indikatörünü yani cursoru başa konumlandırır dosyada.

%f float değeri okuyacağımı gösterir yukarıdaki kodda, &i ile de değişkenin adresini veriyoruz. Ampersant ile kısaca adres bilgisi verilir.

%d %i integer

%u işaretli yani pozitif tam sayı

%o oktal sayı

%x hexadecimal bir sayı

%c karakter

%s karakter veya string dizisi

%f float

%e double

%g %G double veya float

%% yüzde karakteri yazmaya yarar.

Sqrt(2ç0) ile karakök alınır.

AKREP YELKOVAN ÖRNEĞİ:

```
2-dizi-olusturma-ve-donguler x
1 #!/ bin/ bash
2
3 declare -a dizi
4 j=0
5 for i in $*
6 do
7     echo $i
8     dizi[j]=$i
9     ((j+=1))
10 done
11
12 k=0
13 while [ $k -lt ${#dizi[*]} ]
14 do
15     printf "%s" "${dizi[k]}"
16     ((k+=1))
17 done
18 printf "\n"
19
20 ilkeleman=$1
21 ilkelemanboy=${#ilkeleman}
22 echo $ilkeleman
23 echo $ilkelemanboy
24
```

DİZİ OLUŞTURMA VE DÖNGÜLER:

```
celal5 x 1-akrep-yelkovan-aci-hesaplama.c x
1 #include <stdio.h>      Yelkovan ilerledikçe akrepçe ilerler. buna hesaplama yapmamız lazım.
2 #include <string.h>
3 #include <stdlib.h>      Atoi tanımı için include yapılır stdlib,
4 int main(int argc, char* argv[])
5 {
6     if(argc==3)
7     {
8         double akrep=atoi(argv[1])*12; atoi, string türünü int çeviriyor
9         double yelkovan=atoi(argv[2])*60;
10        double aciclakrep=0;
11        double acictiyelkovan=0;
12        acictiyelkovan+=(yelkovan)*6; tüm saat 360 derecedir, aslında yelkovani 6 ile çarpınca aç buluyoruz, mesela 15:00 için 6 ile çarparsak 90 buluyoruz. yani sağa dönük 90 derec
13        aciclakrep+=akrep*30+((yelkovan*6))*30/360; her 1 saatlik ara 30 derece olduğu için bu işlemi yaptık.
14        double fark=acictiyelkovan-aciclakrep;
15        if(fark<0)        aç farkını bulmak için, biri diğerinden ileride ise -1 ile çarpıyoruz ki pozitif olsun.
16        fark=fark*-1;
17        if(fark>180)
18        fark=360-fark;
19        printf("Aradaki Fark:%.2f\n",fark);
20    }
21    else
22        printf("Akrep ve Yelkovan Değerlerini Giriniz\n");
23    return 0;
24 }
```

BU GÜN OLUŞTURULAN DOSYALAR:

```
3-bugun-olusturulan...alar-dizin-argumanli x
1 dizin=$1
2 if [[ $dizin != "" ]] arguman boş mu dolu mu
3 then
4 if [[ -d $dizin ]] dizi n mi
5 then
6 cd $dizin
7 else
8 echo "Dizin Değil"
9 exit 1 hata varsa işlemi burada sonlandırıyor
10 fi
11 fi
12 s=0
13 tarih=$(date) bash kodundan gelen date i tarih e atadık
14 ay=${tarih:4:3} 4 satır sağa gidip ayı yakaladı
15 gun=${tarih:8:2} 8 den sonra 2 tane gitti ve günü yakaladı
16 ls -al > temp_file_bugun tempfile adında bir dosya oluşturdu ve bulunduğumuz dizine yazıldı
17 while read a1 a2 a3 a4 a5 a6 a7 a8 a9 a1 e okuma yazma izinlerini, a2 ye root falan vs sırayla içlerine atıyor.
18 do
19 if [[ $a6 == $ay && $a7 == $gun && $a9 != "temp_file_bugun" ]] günümüzdeki tarih değerleri ile dosyaları karşılaştırıyor. ve kod sırasında yazdığımız "temp_file_bugun" dosyası
20 then
21 s=$((s+1)) # s="expr "$s"+"$a5" | bc"
22 echo $a9 dosyanın ismini yazdırıyoruz
23 # printf "%s %s\n" "$a5 $a9"
24 fi
25 done < temp_file_bugun Bu dosyayı döngünün içine satır bazlı olarak veriyor, her 1 turda 1 satır çekecek
26 rm temp_file_bugun bizim oluşturduğumu geçici dosyayı siliyoruz.
```

```
neo@ubuntu:~$ bash 3-bugun-olusturulan-dosyalar-dizin-argumanli zczczczx
Dizin Değil
neo@ubuntu:~$ bash 3-bugun-olusturulan-dosyalar-dizin-argumanli
.
1-akrep-yelkovan-aci-hesaplama.c
1-akrep-yelkovan-aci-hesaplama.c-
.bash_history
saatcl
neo@ubuntu:~$ bash 3-bugun-olusturulan-dosyalar-dizin-argumanli Downloads/
.
1-akrep-yelkovan-aci-hesaplama.c
neo@ubuntu:~$
```

DOSYA BOYUTU HESAPLAMA

```
3-bugun-olusturulan...alar-dizin-argumanli x 4-dosya-hesap-baska x
1 s=0
2 ls -al > list
3 while read a1 a2 a3 a4 a5 a6 a7 a8 a9
4 do
5 if [[ ( -f $a9 ) && $a5 != 0 ]] Dosya mı değil mi onu kontrol ediyoruz, yani dosyası al
6 then
7 s=$((s+$a5)) s="expr "$s"+"$a5" | bc" ilk turda boyutları toplamış ve dönüştürme işlemi yapmış bc ile, ve s değişkenine atmış
8 printf "%s %s\n" "$a5 $a9" yakalanan dosyanın ismini ve boyutunu yazdırdım s ler string demek değişken olan s ile ilgisi yok
9 fi
10 done < list
11 rm list
12 echo "Total size: $s byte"
```

```
neo@ubuntu:~$ ls -al
total 188
-rwxr-xr-x 1 neo neo 8646 Jan 12 04:47 saatcl
-rwxr-xr-x 1 neo neo 278 Jan 16 2019 soru2.sh-
-rwxr-xr-x 2 neo neo 4096 Dec 7 2018 Templates
-rwxr-xr-x 2 neo neo 4096 Dec 7 2018 Videos
-rw-r--r-- 1 neo neo 51 Jan 24 2020 .Xauthority
-rw-r--r-- 1 neo neo 108 Jan 24 2020 .xsession-errors
-rw-r--r-- 1 neo neo 828 Jan 24 2020 .xsession-errors.old
neo@ubuntu:~$ ls -al
total 188
-rwxr-xr-x 15 neo neo 4096 Jan 12 05:22 .
-rwxr-xr-x 3 root root 4096 Nov 30 2018 ..
-rw-r--r-- 1 neo neo 469 Jan 12 04:47 1-akrep-yelkovan-aci-hesaplama.c
-rw-r--r-- 1 neo neo 491 Jan 12 04:45 1-akrep-yelkovan-aci-hesaplama.c-
-rw-r--r-- 1 neo neo 489 Jan 11 15:46 1-akrep-yelkovan-aci-hesaplama.cpp-
-rw-r--r-- 1 neo neo 244 Jan 11 15:50 2-dizi-olusturma-ve-donguler
-rw-r--r-- 1 neo neo 419 Jan 12 05:19 3-bugun-olusturulan-dosyalar-dizin-argumanli
-rw-r--r-- 1 neo neo 392 Jan 12 05:19 3-bugun-olusturulan-dosyalar-dizin-argumanli-
-rw-r--r-- 1 neo neo 213 Jan 11 16:30 4-dosya-hesap-baska
-rw-r--r-- 1 neo neo 391 Jan 11 16:33 5-dosyadan-farkliokuma.cpp
-rw-r--r-- 1 neo neo 696 Jan 11 16:38 6-dosyada-kelime-arama.cpp
-rw-r--r-- 1 neo neo 153 Dec 20 2018 awkkodu-
-rw-r--r-- 1 neo neo 3494 Jan 12 05:20 .bash_history
-rw-r--r-- 1 neo neo 220 Nov 30 2018 .bash_logout
-rw-r--r-- 1 neo neo 3637 Nov 30 2018 .bashrc
-rwxr-xr-x 14 neo neo 4096 Jan 28 2019 .cache
-rw-r--r-- 1 neo neo 401 Nov 24 01:24 celal2
-rw-r--r-- 1 neo neo 385 Nov 2 2015 celal3
-rw-r--r-- 1 neo neo 589 Dec 15 04:37 celal4
-rw-r--r-- 1 neo neo 601 Dec 15 04:36 celal4-
-rw-r--r-- 1 neo neo 247 Dec 15 05:21 celal5
-rw-r--r-- 1 neo neo 247 Dec 15 05:21 celal5-
```

```
neo@ubuntu:~$ bash 4-dosya-hesap-baska
469 1-akrep-yelkovan-aci-hesaplama.c
491 1-akrep-yelkovan-aci-hesaplama.c-
489 1-akrep-yelkovan-aci-hesaplama.cpp-
244 2-dizi-olusturma-ve-donguler
419 3-bugun-olusturulan-dosyalar-dizin-argumanli
392 3-bugun-olusturulan-dosyalar-dizin-argumanli-
213 4-dosya-hesap-baska
391 5-dosyadan-farkliokuma.cpp
696 6-dosyada-kelime-arama.cpp
153 awkkodu-
3494 .bash_history
220 .bash_logout
3637 .bashrc
401 celal2
385 celal3
589 celal4
601 celal4-
247 celal5
247 celal5-
88 cikttcl
25 .dmrc
54 dosya-
229 dosya1-
1272 .ICEAuthority
675 .profile
8646 saatcl
278 soru2.sh-
51 .Xauthority
108 .xsession-errors
828 .xsession-errors.old
Total size: 6032 byte
neo@ubuntu:~$
```

DOSYA OKUNABİLİYOR MU:

```
3-bugun-olusturulan...alar-dizin-argumanli x 4-dosya-hesap-baska x 5-dosyadan-farkliokuma.c x
1 #include <stdio.h>
2
3 int main(int argc, char* argv[])
4 {
5     FILE *fpr;
6
7     char str[10];
8     fpr = fopen(argv[1], "r"); dosyayi okuma modunda açtım
9
10
11     if (fpr == NULL)
12     {
13         puts("Dosyayı açarken problem oldu");
14     }
15
16     while(1)
17     {
18         if(fgets(str, 10, fpr) == NULL)
19             break;
20         else
21         {
22             printf("%s", str);
23         }
24     }
25
26     fclose(fpr);
27     return 0;
28 }
```

KELİMEYİ ARAYARAK KAÇ TANE OLDUĞUNU BULAN PROGRAM:

```
6-dosyada-kelime-arama.cpp x 7-dosyada-kelime-bul-degistir.cpp x fork.c x
1 #include <stdio.h>
2 #include <string.h>
3 #include <stdlib.h>
4 int main(int argc, char* argv[])
5 {
6     FILE *fp1;
7     int i, toplam=0, sıra=0, konum, bitti=0; "sıra" dosyadaki cursor pozisyonunu tutmak için, "konum" dosyada hareket etmek için kullanılan bildi, dosya sonuna gelince "bitti"
8     char c;
9     char *aranan; char tipinde pointer değişken tanımladık. (c de dinamik bir dizi oluşturabilmek için lazım çünkü ilkeldir C dili
10    aranan=(char*) malloc (sizeof(argv[1])); aranan değişkenine, char* ile bellekteki boyutunu ve tipini(string) ve malloc fonk ile boyu tunu veriyoruz, argv[1] ile aranan
11    strcpy(aranan,argv[1]);
12    fp1= fopen (argv[2], "r");
13    while(1) dosya olana kadar fp1 ile aranacak olan dosyayı okuma modunda açıp alıp içine attık
14    {
15        toplam=0;
16        for(i=0;i<strlen(aranan);i++) kelimenin boyu kadar dönüyoruz
17        {
18            c = fgetc(fp1);
19            sıra++; en son nerede kaldığımızı tutar cursor olarak
20            if(c==EOF) dosya sonuna geldik mi? geldik ise bitti 1 setle ve while ı sonlandır.
21            bitti=1;
22            else
23            {
24                if(c==aranan[i])
25                {
26                    toplam++;
27                    else
28                    break;
29                }
30            }
31            if(toplam==strlen(aranan)) eğer toplam değeri aranana eşitse bulduk demektir.
32            {
33                konum=sıra-strlen(aranan)+1;
34                printf("Bulundu %d\n",konum);
35            }
36            if(bitti==1)
37            break;
38        }
39        fclose(fp1);
40        return 0;
41    }
```

```
neo@ubuntu: ~
neo@ubuntu:~$ gcc -o arabul 6-dosyada-kelime-arama.cpp
neo@ubuntu:~$ ./arabul deneme 27ekim
Bulundu 1 1 tane bulundu
Bulundu 126 126 bYTE olarak bulundu
neo@ubuntu:~$ ./arabul Z1 27ekim
Bulundu 17
Bulundu 19
Bulundu 46
Bulundu 48
Bulundu 52
Bulundu 54
neo@ubuntu:~$
```

KOD ÖRNEKLERİ:

Dosyadaki aranan kelimeyi bularak ondan kaç tane olduğunu veren program:

```
6-dosyada-kelime-arama.cpp x 7-dosyada-kelime-bul-degistir.cpp x fork.c x
1 #include <stdio.h>
2 #include <string.h>
3 #include <stdlib.h>
4 int main(int argc, char* argv[])
5 {
6
7     FILE *fp1;
8     int i, toplam, sıra=0, konum, bitti=0;
9     char c;
10    char *aranan;
11    aranan=(char*) malloc (sizeof(argv[1]));
12    strcpy(aranan,argv[1]);
13    fp1= fopen (argv[2], "r");
14    while(1)
15    {
16        toplam=0;
17        for(i=0;i<strlen(aranan);i++)
18        {
19            c = fgetc(fp1);
20            sıra++;
21            if(c==EOF)
22            {
23                bitti=1;
24            }
25            else
26            {
27                if(c==aranan[i])
28                {
29                    toplam++;
30                }
31                else
32                {
33                    break;
34                }
35            }
36        }
37        if(toplam==strlen(aranan))
38        {
39            konum=sıra-strlen(aranan)+1;
40            printf("Bulundu %d\n",konum);
41        }
42        if(bitti==1)
43        {
44            break;
45        }
46    }
47    fclose(fp1);
48    return 0;
49 }
```

Program parametreler üzerinden çalışıyor, ana programımıza 2 tane parametre gönderiyoruz. 1. parametre dosyada aranacak kelimeyi 2. si ise hangi dosyada arayacağımızı söylüyor.

FILE *fp1; file içinde bir değişken tanımladık(pointer tipinde)

int i, toplam, sıra=0, konum, bitti=0; "sıra" dosyadaki cursor pozisyonunu tutmak için, "konum" dosyada hareket etmek için kullanılan bildi, dosya sonuna gelince "bitti"

char c;

char *aranan; char tipinde pointer değişken tanımladık. (c de dinamik bir dizi oluşturabilmek için lazım çünkü ilkeldir C dili

aranan=(char*) malloc (sizeof(argv[1])); aranan değişkenine, char* ile bellekteki boyutunu ve tipini(string) ve malloc fonk ile boyu tunu veriyoruz, argv[1] ile aranan

strcpy(aranan,argv[1]); aranan icine aktardık

fp1= fopen (argv[2], "r"); değer, kısaca buranın her yerine kullanıcının girdiği aranacak kelimenin boyu kadar bellekte yer ayırdık.

while(1) dosya olana kadar fp1 ile aranacak olan dosyayı okuma modunda açıp alıp içine attık

toplam=0;

for(i=0;i<strlen(aranan);i++) kelimenin boyu kadar dönüyoruz

c = fgetc(fp1);

sıra++; en son nerede kaldığını tutar cursor olarak

if(c==EOF) dosya sonuna geldik mi? geldik ise bitti 1 setle ve while ı sonlandır.

bitti=1;

else

{

if(c==aranan[i])

toplam++;

çekersek ve hepsi sırayla eşit olursa bu kelimeyi bulmuşuz demektir.

else

break;

}

if(toplam==strlen(aranan)) eğer toplam değeri aranana eşitse bulduk demektir.

{

konum=sıra-strlen(aranan)+1;

printf("Bulundu %d\n",konum); tamamen çıkartınca kelimenin ilk karakterine denk gelsin

}

if(bitti==1)

break;

}

fclose(fp1);

return 0;

```
neo@ubuntu: ~
neo@ubuntu:~$ gcc -o arabul 6-dosyada-kelime-arama.cpp
neo@ubuntu:~$ ./arabul deneme 27ekim
Bulundu 1 1 tane bulundu
Bulundu 126 126 byte olarak bulundu
neo@ubuntu:~$ ./arabul 21 27ekim
Bulundu 17
Bulundu 19
Bulundu 46
Bulundu 48
Bulundu 52
Bulundu 54
neo@ubuntu:~$
```

Dosyadaki kelimeleri bulan ve deęiřtiren program:

```
6-dosyada-kelime-arama.cpp x 7-dosyada-kelime-bul-degistir.cpp x fork.c x 27ekim x
5 int main(int argc, char* argv[])
6 {
7     FILE *fp1,*fp2;
8     int i,toplam,sira=0,konum,bitti=0;
9     char c;
10    char *aranan;
11    char *degistirilecek;
12    aranan=(char*) malloc (sizeof(argv[1]));
13    degistirilecek=(char*) malloc (sizeof(argv[2]));
14    strcpy(aranan,argv[1]);
15    strcpy(degistirilecek,argv[2]);
16    fp1= fopen (argv[3], "r+");
17    fp2= fopen ("yenidosyamiz","w"); deęistirilmiř řeklinde olan dosyayı iinde tutacak dosya
18    while(1)
19    {
20        toplam=0;
21        int ictur=0;
22        for(i=0;i<strlen(aranan);i++) burası dosyada kelime aramayla aynı soldaki for ds.
23        {
24            c = fgetc(fp1);
25            sıra++; ictur++;
26            if(c==EOF)
27                bitti=1;
28            else
29            {
30                if(c==aranan[i])
31                    toplam++;
32                else
33                    break;
34            }
35        }
36        if(bitti==1)
37            break;
38        if(toplam==strlen(aranan)) aranan kelime bulunursa
39        {
40            konum=sıra-strlen(aranan)+1;
41            printf("Bulundu %d\\n",konum); bulunduęu yeri yazdım
42            fputs(degistirilecek,fp2); kelimenin bulunduęu yere yazılacaktır olarak bu iřlemi yaşıyoruz
43        }
44        else eęer aranan kelime bulunamazsa
45        {
46            konum=sıra-ictur;
47            //printf("sıra=>%d ictur=>%d konum=>%d",sıra,ictur,konum);
48            fseek(fp1,konum,SEEK_SET);
49            for(i=0;i<ictur;i++)
50            {
51                char p=fgetc(fp1);
52                //printf("%c\\n",p);
53                fprintf(fp2,"%c",p);
54            }
55        }
56    }
57    fclose(fp1); dosya aık diyip hata verir o yzden bu iřlemi yaotık
58    fclose(fp2);
59    rename("yenidosyamiz",argv[3]);yenidosyamiz isimli dosyanın ismini argv[3] olarak deęiřtiriyoruz, kullanıcı hangi dosyayı aarsa onu yeni dosyamız olarak grmesidir.
60    return 0;
61 }
```

```
neo@ubuntu:~
neo@ubuntu:~$ gcc -o buldegis 7-dosyada-kelime-bul-degistir.cpp
neo@ubuntu:~$ ./buldegis satirın mısram 27ekin
Bulundu 60
Bulundu 133
neo@ubuntu:~$
```

satırmları mısram olarak deęiřtirdi.

Process Management sf 3

Sistem aęrılarını nelerle uęrařır?

- *Bir processin oluřturulması.
- *Bir processi icra edecek řekilde ayarlamak.
- *Bir bařka iřlem bir processin sonlandırılmasını beklemek

- *Processler arası haberleşme
 - *Processi bitirmek
 - *Bir processe sinyaller göndermek.
-

Process ID sadece bir sayıdır getpid() ile ulaşılabilir

ps ile Id li processleri listeleyebiliyoruz.

her processin bir parenti var getppid() ile ulaşılabilir o id ye. (Get process parent id kısaltması)

getpgrp() her process bir grubun üyesidir, bunun ile de group id getirebiliyoruz.

Processlerin Oluşturulması:

pid = fork() ile bir tane yeni çocuk process üretiyor, o çocuk process i o anda koşan bir processin kopyasını oluşturuyor. Parent ve çocuk aynı konumda koşturmaya devam eder.

Aynı değişken değerlerini alarak aynı dosyaları açabiliyorlar.

Tek fark çocuk yeni bir PID değerine sahip olur.

Eğer forkdan dönen değer 0 ise çocuk process anlamına gelir.

Eğer parentin içerisinde ise childin PID değeri döner.

Pid = wait(&status) ile, çağıran processin icrasını durduruyor, yani parent processin icrası durdurulur. Kısaca, parentin icrasını, üretilen çocuklardan herhangi birinin icrası sonlanırsa onun process id sini almış oluyoruz.

Pid = wait (pid,&status,options)Çağırılan processin icrasını, belirli bir çocuk process sonlanırsa durdurur.(spesifik bir çocuk)

```
pid = fork();
if( pid == 0)
    execl("./program", "program",
        arg1, NULL);
else
    pid = wait(& status);continue
    execution
```

yandaki kod parçasında, process çocuk ise programın versiyonunu oluştur, eğer değilse çocuk sonlanıncaya kadar bekle.

Exit(status) açılmış tüm processleri kapatır.

Bütün dosya açıklamalarını da kapatır.

Parent processe de processin durumunu dönderir.

SİNYAL İSİMLERİ:

INT Kesme isteği yollar

KILL Processi zorla bitiriyoruz

ALRM Alarm

TERM Yakalanabilir bir icra sonucu düzgün bir giriş

QUIT Çıkış

ABRT iptal

HUP Asılı kalmak

Sinyaller, sinyal tutucular tarafından yakalanır. (Signal Handler)

Sf12-----

Sending signals to process

SIGINT adında bir

- Send a signal using kill function
 - `retval = kill(pid, signal)`
- Example
 - `kill(SIGINT, 1234);`
 - Sends an interrupt (signal) of type SIGINT
 - Does not block sending process
 - The process, whose ID is 1234, gets the signal.

fork.c (-) - gedit



Open Save Undo redo Print Copy Paste Find Replace

6-dosyada-kelime-arama.cpp x 7-dosyada-kelime-bul-degistir.cpp x fork.c x 27ekim x

```
1 #include <stdio.h>
2 #include <unistd.h>
3 #include <stdlib.h>
4 #include <sys/wait.h>
5 int main ( void )
6 {
7     int f;
8     f= fork();
9     if(f==0)
10 {
11 printf("Cocuk : Proses no: %d \n" , getpid() );
12 sleep(10);
13 printf("Cocuk : Anne proses no : %d \n" , getppid() );
14 }
15 else
16 {
17 printf("Anne : Proses no : %d \n", getpid() );
18 printf("Anne : Cocuk proses no : %d \n" , f );
19 printf("Anne : Anne proses no : %d \n" , getppid() );
20 wait(NULL);
21 printf("Anne : Sonlanıyorum . . . \n");
22 exit(0);
23 }
24 return 0;
25 }
```