

Biyoinformatik · Uygulama notları

Biyoinformatik için Terminalin Temelleri

Linux ve WSL2'den Bash, Conda ve Python ile
Tekrarlanabilir Küçük Uygulamalara

Bu not, terminal ve programlama deneyimi sınırlı öğrenciler için hazırlanmıştır. Bir komutun hangi ortamda çalıştığını, hangi dosyaları etkilediğini ve sonucunun nasıl denetleneceğini adım adım açıklar. Doğrudan Linux kurulu bir bilgisayar ile Windows üzerinde WSL2 kullanan bir bilgisayar, aynı Bash ve Python uygulamalarında buluşur. Kaynak sınırları, yazılım ortamları ve dosya yerleşimi, yapılan işlemin gerekçesiyle birlikte ele alınır. Küçük, sentetik bir dizi tablosu üzerinden dosya incelemeden yeniden çalıştırılabilir bir analize geçilir.

Kod ve yazılım ortamları:
main_scripts/terminal_basics/

Tuna Birgün
github.com/tunabirgun

31 Ekim 2026

İçindekiler

1 Temel kavramlar **2**
2 Linux ve WSL2 kurulumu **3**
3 WSL2'ye ayrılan RAM ve işlemci kaynakları **4**
4 Dosya sistemi ve temel terminal kullanımı **6**
5 Komutları birleştirme ve Bash betikleri **8**
6 Conda ekosistemi ve proje ortamları **11**
7 Python'a temel giriş **14**
8 Performans ve I/O darboğazları **16**
9 Bütünleştirici uygulama ve tekrarlanabilirlik **18**
Kaynakça **22**

1 Temel kavramlar

Bir analiz aracını kullanmak, yalnızca komutun adını bilmek değildir. Aracın hangi girdiyi okuduğunu, çıktıyı nereye yazdığını ve başarısız olduğunda nasıl haber verdiğini de bilmek gerekir. Terminal bu ilişkileri görünür kılar. Örneğin bir dizi tablosundan üç kayıt okumak, kayıtların uzunluklarını hesaplamak ve sonucu yeni bir dosyaya yazmak ayrı işlemlerdir; bunları açıkça tanımlamak, aynı analizi başka bir gün yeniden yapabilmenin başlangıcıdır.

1.1 Ekrandaki pencerenin arkasında ne var?

İşletim sistemi, donanımı ve programların kullandığı kaynakları yönetir. Bunun merkezindeki **çekirdek**, işlemci zamanının paylaşılması, bellek ve dosya erişimi gibi işleri düzenler. Linux teknik olarak çekirdeğin adıdır. Ubuntu gibi bir **dağıtım**, bu çekirdeği komut araçları, paket yöneticisi ve başka yazılımlarla birlikte sunar. Bu notta Linux sözü, bağlama göre bu çalışma ortamının bütünü için de kullanılır.

Terminal, yazdığınız metni ve programların yanıtlarını gösteren arayüzdür. **Kabuk** ise komut satırını yorumlayan programdır. Kabuk bir komutu okuyabilir, değişkenleri genişletebilir ve başka bir program başlatabilir. Çalışmakta olan bir program örneğine **süreç** denir; her sürecin bir kimliği, kullandığı kaynaklar ve bir çalışma dizini vardır. Aynı Python betiğini iki kez başlatmak, iki ayrı süreç oluşturur.

Komut, kabuktan bir işlem yapmasını isteyen ifadedir. Komutun adı bir kabuk yerleşiğini veya diskteki bir programı gösterebilir. Örneğin `cd` çalışma dizinini kabuğun içinde değiştirir; `python` ise çoğu kurulumda ayrı bir program başlatır. Kabuk, program adlarını `PATH` adlı ortam değişkenindeki dizinlerde arar. Bu yüzden aynı `python` sözcüğü, etkinleştirilen yazılım ortamına göre farklı bir yorumlayıcıyı çalıştırabilir.[1]

1.2 Bash, PowerShell ve Windows Terminal

Bash, Linux dağıtımlarında yaygın kullanılan bir kabuk ve betik dilidir. **PowerShell**, Windows yönetiminde sık kullanılan, başka işletim sistemlerinde de çalışabilen ayrı bir kabuk ve dildir. PowerShell komutları arasında nesneler aktarılabilir; bu nottaki Bash boruları ise programların metin veya bayt çıktıları üzerinde çalışır. Benzer görünen komut adları, seçeneklerinin de aynı olduğunu göstermez.[2]

Windows Terminal, bu kabukları sekmeler içinde açan uygulamadır. Bir sekmede PowerShell, diğerinde Ubuntu'nun Bash kabuğu bulunabilir. Pencerenin aynı görünmesi, iki sekmenin aynı dili veya dosya sistemini kullandığı anlamına gelmez.[3]

Uygulama notu

Bu notta **WINDOWS / POWERSHELL** kutularını Windows'ta PowerShell sekmesinde; **LINUX / BASH** kutularını doğrudan Linux'ta veya WSL2 içindeki Ubuntu sekmesinde çalıştırın. **PYTHON** kutuları Python kodudur; bunları Bash'e doğrudan yapıştırmayın. Çıktı ve yapılandırma kutuları çalıştırılacak komut değildir. Yönetici yetkisi gereken Windows adımları ayrıca belirtilir. Komutların başına ekranınızda görünen `$` veya `PS>` istemlerini eklemeyin.

1.3 Bir komutu okuyabilmek

Aşağıdaki komutta `ls` programın adı, `-l` uzun biçimde listeleme seçeneği, `/tmp` ise üzerinde işlem yapılacak dizindir. Bir **argüman**, programa iletilen bilgi parçasıdır; seçenekler de argümanların bir türüdür.

LINUX / BASH

```
ls -l /tmp
```

Çıktının dosya adları bilgisayara göre değişir. Burada beklenen, `/tmp` içeriğinin ayrıntılı bir listesidir. Bir komutun çıktısı boş olabilir; bu tek başına hata değildir. İşlemin sonucunu dosyalar, açıklayıcı hata iletileri ve ileride göreceğimiz çıkış kodlarıyla değerlendireceğiz.

Düşünme alıştırması. Windows Terminal'de Ubuntu sekmesini kapatıp PowerShell sekmesini açınca neden Linux'taki `cd /home` komutunu aynen sürdürmeyiz? Çünkü değişen yalnızca görünüm değildir; komutu yorumlayan kabuk ve başlangıçta erişilen dosya sistemi de değişmiştir. Linux oturumuna geçişi açıkça yapmak gerekir.

2 Linux ve WSL2 kurulumu

Doğrudan kurulumda Linux çekirdeği bilgisayarın donanımı üzerinde çalışır. WSL2'de ise Linux çekirdeği, Windows'un yönettiği hafif bir sanal makine içinde çalışır. Ubuntu'nun kullanıcı araçları her iki yolda da kullanılabilir; WSL2'de Windows ile Linux aynı fiziksel RAM'i, işlemciyi ve diski paylaşır. WSL1 farklı bir mimaridir; bu notun hedefi WSL2'dir.[4]

2.1 Doğrudan Linux kurulumu

Bilgisayarında zaten desteklenen bir Ubuntu sürümü olan öğrenci yeniden kurulum yapmadan doğrulama adımına geçebilir. Yeni kurulum için Ubuntu'nun resmî masaüstü kurulum kılavuzundaki sıra izlenir: Uygun mimarideki ISO dosyasını indirme ve doğrulama, kurulum USB'sini hazırlama, USB'den başlatma, canlı oturumda donanımı deneme, disk düzenini seçme ve kullanıcı oluşturma.[5]

USB'ye kurulum görüntüsü yazmak USB'nin içeriğini değiştirir. Disk bölümleme de mevcut verilere zarar verebilir; kurulumdan önce yedek alın ve hedef diski doğrulayın. Windows'u korumak isteyen öğrenci, diski silen seçeneği seçmemelidir. Çift açılış, BitLocker ve disk bölümleri konusunda deneyim yoksa kurulumu eğitmen eşliğinde yapmak uygundur. Burada tek bir disk düzeni bütün öğrencilere dayatılmaz. Desteklenen bir LTS sürümüyle başlayın; dersin paketleri için seçilen sürümü kaydedin.

2.2 Windows'ta WSL2 ve Ubuntu

Microsoft'un kolay kurulum yolu Windows 11 veya Windows 10 sürüm 2004/build 19041 ve sonrasında bulunur; bu tarihsel alt sınır, eski bir Windows sürümünün bugün desteklendiği anlamına gelmez. Güncel ve desteklenen Windows kurulumunu kullanın. Donanım sanallaştırması UEFI/BIOS'ta açık olmalıdır. Ubuntu'nun WSL kılavuzu da sanallaştırma ve Virtual Machine Platform gereksinimini açıklar.[6, 7]

Başlat menüsünden PowerShell'i **yönetici olarak** açın. Önce mevcut duruma bakın; kurulu bir Ubuntu'yu silmek veya yeniden kurmak gerekmez. Yeni kurulum için resmî komut, gerekli Windows bileşenlerini etkinleştirir ve dağıtımı kurar.

WINDOWS / POWERSHELL

```
wsl --list --verbose
wsl --list --online
wsl --install -d Ubuntu
```

İlk komut WSL henüz yoksa kurulum gerektiğini bildirebilir. Son komuttan sonra istenirse Windows'u yeniden başlatın. Çevrim içi listeden farklı bir sürümlü dağıtım seçtiyseniz sonraki komutlarda da listede görünen tam adı kullanın. `Ubuntu` adı gelecekte başka bir varsayılan sürümü gösterebilir; gerçek sürümü Linux içinden okuyacağız.

Ubuntu'yu Başlat menüsünden açın. İlk açılışta Linux kullanıcı adını ve parolasını oluşturun. Bu hesap Windows hesabından ayrıdır. Parola yazarken ekranda yıldız veya karakter görünmemesi normaldir. Günlük analizleri bu normal kullanıcıyla yapın; `sudo` yalnızca gerekli sistem işlemlerine geçici yetki verir.[8]

2.3 Kurulumu doğrulama

Aşağıdaki Windows komutlarını normal PowerShell oturumunda çalıştırın. `--version`, WSL yazılımının sürümünü; `--list --verbose` ise her dağıtımın WSL1 mi WSL2 mi kullandığını gösterir. Bu iki sürüm kavramı aynı değildir.[9]

WINDOWS / POWERSHELL

```
wsl --version
wsl --status
wsl --list --verbose
wsl -d Ubuntu
```

Dağıtım satırının `VERSION` sütununda `2` bulunmalıdır. `Stopped` yazması bozuk kurulum anlamına gelmez; dağıtım henüz çalışmıyor olabilir. Son komut Linux oturumunu açar. Bundan sonra aşağıdaki komutlar **Ubuntu/Bash içinde** çalışır; doğrudan Linux kullananlar da aynı kontrolü yapar.

LINUX / BASH

```
whoami
pwd
cat /etc/os-release
uname -r
uname -m
bash --version
```

`whoami` kullanıcıyı, `pwd` çalışma dizinini, `/etc/os-release` dağıtımı, `uname -r` çekirdeği gösterir. `uname -m` sonucu `x86_64` veya `aarch64` gibi bir mimari adıdır. Bu değer, ileride indireceğimiz Linux kurucusuyla uyumlu olmalıdır.

WSL2 yerine WSL1 görünüyorsa, önce çalışmaları yedekleyip Microsoft'un dönüşüm yönergesini izleyin; dönüşüm sürebilir. Yeni kurulacak dağıtımlar için varsayılanı değiştirmek, mevcut dağıtımı dönüştürmez. Bu ayrımı gösteren komutlar PowerShell'de çalışır.[9]

WINDOWS / POWERSHELL

```
wsl --set-default-version 2
wsl --set-version Ubuntu 2
```

Kontrol sorusu. `uname -r` ile Ubuntu sürümünü öğrenebilir miyiz? Hayır; bu çekirdek sürümüdür. Dağıtım sürümü için `/etc/os-release`, Windows tarafındaki WSL paketi için `wsl --version` gerekir.

3 WSL2'ye ayrılan RAM ve işlemci kaynakları

Paket kurmaya başlamadan önce WSL2'nin kaynak sınırını belirlemek gerekir. Bir bilgisayarın fiziksel RAM'i ile Linux'un kullanmasına izin verilen bellek aynı sayı olmayabilir. Windows, tarayıcı ve diğer açık uygulamalar da belleğe ihtiyaç duyar. Linux'a bütün RAM'i vermek, toplam kullanılabilir belleği artırmaz; Windows tarafındaki baskıyı artırabilir.

3.1 Önce kapasiteyi ve mevcut kullanımı ayırın

Windows Görev Yöneticisi'nin Performans bölümünde toplam RAM'i, o anki kullanımı, kullanılabilir belleği ve mantıksal işlemci sayısını inceleyin. Toplam kapasite sabittir; kullanılabilir miktar açık uygulamalara göre değişir. WSL2 için seçtiğiniz **bellek üst sınırı**, analizlere ayrılmış ve baştan bütünüyle tüketilen sabit bir blok olarak düşünülmemelidir.

Güncel Microsoft belgesinde varsayılan bellek sınırı Windows RAM'inin yarısıdır; işlemci sayısı varsayılan olarak Windows'taki mantıksal işlemci sayısıyla aynıdır. Bunlar sizin iş yükünüz için önerilen değerler değildir.[10] Örneğin 16 GB RAM'li bir bilgisayarda tarayıcı ve başka ders yazılımları açıkken, WSL için düşünülen 8 GB sınırını gerçek kullanım altında gözlemlemek gerekir. 8 GB toplam RAM'i olan bir bilgisayara aynı ayar uygun değildir. Büyük bir analiz tek başına sınırı aşıyorsa daha çok swap eklemek yerine girdiyi küçültmek, eş zamanlı işleri azaltmak veya uygun bir sunucu kullanmak gerekebilir.

3.2 WSL Settings ve .wslconfig

Microsoft, uygun sürümlerde Başlat menüsündeki **WSL Settings** uygulamasını önerir. Bellek, işlemci ve swap alanlarını bu arayüzden düzenleyebilirsiniz. Arayüzdeki alanlar sürüme göre değişebilir. Metin üzerinden yapılandırma için Windows kullanıcı profilindeki `.wslconfig` dosyası kullanılır; Linux içindeki `/etc/wsl.conf` farklı amaçlıdır. `.wslconfig` ayarları WSL2 dağıtımlarının ortak sanal makine kaynaklarını etkiler.[10]

Aşağıdaki PowerShell komutu dosyayı Not Defteri'nde açar. Dosya zaten varsa içeriğini önce okuyun ve kopyasını saklayın; başka ayarları silmeden ilgili değerleri düzenleyin. Dosya adı `.wslconfig.txt` olmamalıdır.

WINDOWS / POWERSHELL

```
notepad "$env:USERPROFILE\.wslconfig"
```

Aşağıdaki içerik, **16 GB RAM ve en az 8 mantıksal işlemcili varsayımsal bir ders bilgisayarı** için ayar sözdizimini gösterir; ölçülmüş ideal değer veya bütün öğrencilere öneri değildir.

BEKLENEN ÇIKTI / DOSYA İÇERİĞİ

```
[wsl2]
memory=8GB
processors=4
swap=2GB
```

`memory` Linux sanal makinesinin RAM üst sınırıdır. Sınırı artırmak, belleğe sığmayan bir işe yardımcı olabilir; ancak Windows için daha az hareket alanı bırakır. `processors` kullanılabilir mantıksal işlemcileri sınırlar. Dört işlemci görmek, her programın dört işlemciyi otomatik kullanacağı anlamına gelmez; analiz aracının iş parçacığı seçimi ayrıca yapılır. Daha çok iş parçacığı da her zaman daha kısa süre sağlamaz, bazen bellek tüketimini artırır.

Swap, bazı bellek sayfalarının diskte tutulmasına ayrılan alandır. Fiziksel RAM ile eşdeğer değildir. Aktif verilerin sürekli RAM ile disk arasında taşınması büyük gecikmelere yol açabilir. Swap kullanımının sıfırdan büyük olması tek başına sürekli yavaşlamayı kanıtlamaz; zamana yayılan giriş/çıkış etkinliği de incelenmelidir. Çok düşük bellek sınırı veya yetersiz toplam bellek, süreçlerin bellek yetersizliği nedeniyle sonlandırılmasına yol açabilir.

3.3 Ayarı uygulama ve doğrulama

Dosyayı kaydedin, Linux oturumlarındaki işleri ve açık dosyaları güvenli biçimde kapatın. Aşağıdaki komut **bütün çalışan WSL dağıtımlarını** durdurur; devam eden bir analizin ortasında kullanılmaz. Ardından Ubuntu'yu yeniden açın.[9]

WINDOWS / POWERSHELL

```
wsl --shutdown
wsl -d Ubuntu
```

Yeni Linux/Bash oturumunda görülen sınırları kontrol edin.

LINUX / BASH

```
free -h
nproc
swapon --show
```

`free -h` bellek ve swap özetini okunabilir birimlerle verir. `total`, `free` ve `available` farklıdır; `available`, önbellekten geri kazanılabilecek alanı da dikkate alan bir tahmindir. Örnekte işlemci kısıtı uygulanmışsa `nproc` için 4 beklenir; başka süreç kısıtları varsa değer farklı olabilir. Bellek çıktısının birimi ve çekirdeğin ayırdığı alan nedeniyle ekranda tam `8.0Gi` görülmesi şart değildir. Swap kapalıysa `swapon --show` boş çıkabilir.

Beklenen değişiklik görünmüyorsa dosyanın Windows profilinde olduğunu, uzantısını, `[wsl2]` başlığını ve dağıtımın gerçekten WSL2 kullandığını kontrol edin. Sonra kapatma ve yeniden açma adımını doğrulayın. Doğrudan Linux kurulumunda `.wslconfig` kullanılmaz; aynı bellek izleme komutları yine yararlıdır.

Alıştırma. WSL'ye 4 mantıksal işlemci ayırmışken analiz aracını 16 iş parçacığıyla çalıştırmak neden iyi bir başlangıç değildir? Daha çok iş parçacığı ek işlemci yaratmaz; görevler işlemci zamanı için yarışabilir ve daha fazla bellek kullanabilir. Önce küçük bir girdi ve sınırlı eş zamanlılıkla başlayıp ölçmek gerekir.

4 Dosya sistemi ve temel terminal kullanımı

Dosyayı okuyamayan bir komutun ilk sorunu çoğu zaman veri biçimi değildir; yanlış dizinde çalışıyor olabiliriz. Bu nedenle analizden önce nerede olduğumuzu ve hangi dosyayı kastettiğimizi belirleyeceğiz. Bu bölümden itibaren, aksi belirtilmedikçe bütün komutlar doğrudan Linux veya WSL2 içindeki Bash'te çalışır.

4.1 Kök dizin, ev dizini ve yollar

Linux dosya sistemi `/` kökünden başlayan bir ağaçtır. Kullanıcının ev dizini genellikle `/home/kullanici` altındadır. `~`, Bash'te ev dizinini belirtir; `.` bulunduğunuz dizin, `..` onun üst dizinidir. **Mutlak yol** kökten başlar; **göreli yol** çalışma dizinine göre yorumlanır. Örneğin `/home/ogrenci/proje/veri.tsv` mutlak, `proje/veri.tsv` göreli bir yoldur.

Aşağıdaki uygulama, gerçek verilerinizden ayrı bir alanda çalışır. `terminal-pratik` adlı bir dizin zaten varsa yeni bir eğitim dizini adı seçin; eski dosyaların üstüne yazmayın.

LINUX / BASH

```
cd ~
mkdir terminal-pratik
cd terminal-pratik
mkdir veri sonuc betikler
pwd
ls
```

`mkdir` dizin oluşturur. İlk `mkdir` dizin zaten varsa hata verir; bu, çalışmayı durdurup mevcut içeriği incelemek için bir işarettir. `ls` çıktısında `betikler`, `sonuc` ve `veri` görülmelidir; sıralama yerel ayarlara göre değişebilir. Terminalde Tab tuşu yol tamamlama, yukarı ok komut geçmişi için kullanılır. `Ctrl+C` genellikle ön plandaki işi keser; bir dosyayı kaydetme kısayolu değildir.

Dosya adları Linux'ta büyük/küçük harfe duyarlıdır: `Veri.tsv` ile `veri.tsv` farklı adlardır. Boşluklu adlar kullanılabilir ama tırnak gerektirir. Ders uygulamalarında kısa, boşluksuz ve anlamlı adlar seçmek, yazım hatalarını azaltır; gerçek bir örnek kimliğini kolaylık olsun diye değiştirmeyin.

4.2 WSL'de dosyayı nereye koymalı?

WSL2'nin Linux dosya sistemi genellikle sanal disk içinde ext4 kullanır. `~/terminal-pratik` burada bulunur. Buna karşılık `/mnt/c/Users/ogrenci` Windows'un C sürücüsündeki dosyalara erişim yoludur. Her ikisini Bash'ten görebilmek, aynı erişim yolu ve izin davranışına sahip oldukları anlamına gelmez. Microsoft, Linux araçlarıyla işlenen projelerin Linux dosya sisteminde tutulmasını önerir.[11]

WSL kullanan öğrenci Linux'taki mevcut dizini Windows Dosya Gezgini'nde aşağıdaki komutla açabilir. Bu bir Windows programını Linux oturumundan çağırma örneğidir; doğrudan Linux kurulumunda bu komut gerekmez.

LINUX / BASH

```
explorer.exe .
```

Windows Dosya Gezgini'nden `\\wsl.localhost\Ubuntu\home` yolu da kullanılabilir. Dağıtım adı sizde farklıysa yolu değiştirin. Linux dosyalarını dağıtımın Windows uygulama-verisi klasöründen elle düzenlemeyin; desteklenen paylaşım yolunu kullanın. Windows'tan alınan girdiyi Linux proje dizinine kopyalarken

kaynağı koruyun ve kopyanın içeriğini doğrulayın. Bir yolu tırnaklamak boşluk sorununu çözer; dosyanın gerçekten orada olduğunu kanıtlamaz.

4.3 Küçük bir tablo oluşturma ve inceleme

Şimdi üç sentetik kaydın etiketini ve uzunluğunu yazalım. `printf` biçimlendirilmiş metin üretir; `\t` sekme, `\n` satır sonudur. `>` çıktıyı dosyaya yönlendirir ve aynı adlı mevcut dosyayı sıfırlar. Bu komutu yalnızca yeni eğitim dizininde, hedef henüz yokken çalıştırın.

LINUX / BASH

```
printf 'id\tlength\nseq_a\t8\nseq_b\t12\nseq_c\t8\n' > veri/uzunluk.tsv
cat veri/uzunluk.tsv
head -n 2 veri/uzunluk.tsv
wc -l veri/uzunluk.tsv
```

`cat` bütün içeriği, `head -n 2` ilk iki satırı gösterir. Büyük dosyada önce birkaç satıra bakmak daha kullanışlıdır. `wc -l` burada 4 verir: Üç veri satırı ve bir başlık vardır. Satır sayısı örnek sayısıyla kendiliğinden aynı değildir. Sonunda satır sonu olmayan dosyalarda `wc -l` görünür son kaydı da beklediğiniz gibi saymayabilir.[12]

BEKLENEN ÇIKTI / DOSYA İÇERİĞİ

```
id      length
seq_a   8
seq_b   12
seq_c   8
```

Kutudaki sütun boşlukları görsel hizalamadır; gerçek dosyada ayraç sekmedir. `less veri/uzunluk.tsv` ile dosyada gezinebilir, `q` tuşuyla çıkabilirsiniz. Çıkış için `Ctrl+C` yerine programın kendi kısayolunu öğrenmek önemlidir.

4.4 Arama, süzme ve yardım

`grep`, satır içeriğinde örüntü arar; `find` dosya ağacında ad ve tür gibi özellikleri arar. Aşağıdaki ilk arama tam metni arar; ikinci aramadaki yıldızı kabuğun erken genişletmemesi için tırnak kullanılır.

LINUX / BASH

```
grep -F 'seq_b' veri/uzunluk.tsv
find veri -type f -name '*.tsv'
cut -f 2 veri/uzunluk.tsv
```

`grep -F`, örüntüyü düzenli ifade yerine düz metin sayar. Burada `seq_b` satırını bulur; ama bu komut tek başına ilk sütunda tam kimlik eşleşmesi denetimi değildir. `cut -f 2`, sekmeyle ayrılmış ikinci alanı alır ve başlığı da içerir. Alıntı içindeki virgülleri olan genel bir CSV dosyası için böyle basit bölme kuralları uygun olmayabilir; Python'un `csv` modülünü kullanacağız.[13, 14]

Yardım metni, bir seçeneğin hangi sürümde ne yaptığını öğrenmek için ilk başvuru yeridir. `man` sistemde kurulu kılavuzu açar; `help` Bash yerleşikleri içindir.

LINUX / BASH

```
ls --help
man ls
help cd
type cd
command -v python
```

`man` ekranından `q` ile çıkılır. Henüz Python ortamı kurulmadıysa son komut boş kalabilir. Bir programın hangi yoldan seçildiğini bilmek, yanlış ortamdan gelen sürümü ayırt etmeyi sağlar.

4.5 İzinler ve güvenli dosya işlemleri

`ls -l` çıktısındaki `r`, `w` ve `x`; sahip, grup ve diğer kullanıcılar için okuma, yazma ve çalıştırma izinlerini gösterir. Dizin üzerindeki `x`, dizinin içinden geçip altındaki adlara erişebilme anlamı taşır. `chmod u+x betik.sh` yalnızca sahibine çalıştırma izni ekler; izin hatasını gidermek için bütün dosyalara `777` vermek uygun değildir. Windows sürücülerinin WSL içindeki izinleri Windows izinleri ve bağlama ayarlarıyla da ilişkilidir.[12, 15]

Kopyalamadan veya taşımadan önce kaynağı ve hedefi listeleyin. Aşağıdaki örnekte `-i`, hedef varsa üzerine yazmadan önce sorar. Silme örneği yalnızca bu adımda oluşturduğunuz kopya içindir.

LINUX / BASH

```
ls -l veri/uzunluk.tsv
cp -i veri/uzunluk.tsv sonuc/uzunluk-kopya.tsv
cmp veri/uzunluk.tsv sonuc/uzunluk-kopya.tsv
mv -i sonuc/uzunluk-kopya.tsv sonuc/deneme.tsv
rm -i sonuc/deneme.tsv
```

`cmp` eşit dosyalarda sessizdir ve başarı kodu döndürür. `rm`, dosyayı genellikle çöp kutusuna taşımaz; bu nedenle geniş jokerlerle silme yapmayın. `sudo` ile analiz dosyalarını oluşturmak da daha sonra sahiplik sorunlarına yol açabilir. Yetkiyi artırmadan önce yolun, sahipliğin ve asıl gereksinimin doğru olduğunu kontrol edin.

Alıştırma. `veri` dizininin içindeyken `cat veri/uzunluk.tsv` neden bulunamayabilir? Göreli yol artık `veri/veri/uzunluk.tsv` anlamına gelir. `pwd` ile konumu doğrulayın; bu durumda `cat uzunluk.tsv` veya proje köküne dönmek doğru çözümdür.

5 Komutları birleştirme ve Bash betikleri

Tek tek komutlar, ne olduğunu görerek öğrenmeyi kolaylaştırır. Tekrar eden adımlar netleştiğinde bunları bir dosyada saklamak, komut geçmişinden eksik veya yanlış sırada kopyalama riskini azaltır. Bir betik, işlemlerin hangi girdiler üzerinde ve hangi sırayla yapılacağını açıkça tanımlar.

5.1 Üç akış ve çıkış kodu

Bir programın **standart girdisi** (0), **standart çıktısı** (1) ve **standart hatası** (2) vardır. Normal çıktı ile tanı iletilerini ayırmak, veri dosyasının içine hata metni karışmasını önler. `<` dosyayı girdi olarak verir; `>` çıktıyı yazar, `>>` sona ekler; `2>` hata akışını ayrı bir dosyaya gönderir.[16]

`~/terminal-pratik` dizininde aşağıdaki örneği çalıştırın. `wc -l < dosya` biçiminde `wc`, dosya adını değil standart girdiyi okur; bu nedenle çıktı yalnızca sayıdır.

LINUX / BASH

```
cd ~/terminal-pratik
wc -l < veri/uzunluk.tsv
cat veri/yok.tsv > sonuc/okuma.txt 2> sonuc/hata.txt
printf '%s\n' "$?"
cat sonuc/hata.txt
```

İlk sonuç 4'tür. İkinci komutun girdisi bilerek yoktur; hata akışında dosyanın bulunamadığını belirten ileti, hemen sonraki `$?` içinde sıfırdan farklı kod beklenir. Hata metninin dili ve kodun tam değeri araca göre değişebilir. Yönlendirme, program başlamadan yapıldığı için başarısız komut bile boş bir `okuma.txt` oluşturabilir. Dosyanın varlığı, analizin başarıyla tamamlandığını göstermez.

Sıfır çıkış kodu genellikle başarıdır; diğer değerlerin anlamı programa bağlıdır. Örneğin `grep`, eşleşme yoksa 1 döndürür; bu, girdi bozuk demek değildir. `$?` yalnızca son komutun kodudur. Araya başka bir komut koyarsanız önceki sonucu kaybedersiniz.

5.2 Pipe ve adımlar arasındaki bağımlılık

Pipe (boru), bir programın standart çıktısını sonraki programın standart girdisine bağlar. Aşağıdaki işlem başlığı atlar, uzunluk sütununu alır, sayısal sıralar ve aynı değerleri sayar.

LINUX / BASH

```
tail -n +2 veri/uzunluk.tsv | cut -f 2 | sort -n | uniq -c
```

Beklenen sonuç, uzunluğu 8 olan iki ve uzunluğu 12 olan bir kayıttır. `uniq` yalnızca ardışık eşit satırları birleştirdiği için önce sıralama yapılır. Pipe'ı uzatmadan önce her ara çıktıyı ayrı inceleyin; yanlış sütunu seçen bir zincir de hatasız sonlanabilir.

Bash'te borunun çıkış kodu varsayılan olarak son komuttan gelir. `set -o pipefail`, önceki bir adımın başarısızlığını da borunun sonucuna yansıtır. `&&` sonraki komutu yalnızca önceki başarılıysa, `||` ise önceki başarısızsa çalıştırır.[17]

LINUX / BASH

```
set -o pipefail
cat veri/yok.tsv | wc -l
printf '%s\n' "$?"
```

`wc` sıfır yazabilir; bu gerçek veride sıfır kayıt bulunduğunu göstermez. Borunun sıfırdan farklı çıkış kodu ve `cat` hatası, girdi okunamadığını gösterir. Başarı koduyla birlikte içerik denetimine ihtiyaç duymamızın nedeni budur.

5.3 Değişkenler, tırnaklar ve argümanlar

Değişken atamasında eşittirin çevresinde boşluk bulunmaz. Çift tırnak değişkeni genişletirken boşluklu değeri tek argüman olarak korur. Tek tırnak içeriği olduğu gibi tutar.[18]

LINUX / BASH

```
girdi="veri/uzunluk.tsv"
printf '%s\n' "$girdi"
printf '%s\n' '$girdi'
wc -l "$girdi"
```

İlk `printf` yolu, ikincisi `$girdi` metnini yazar. Dosya yollarını `"$girdi"` biçiminde kullanma alışkanlığı edinin. `"~/veri"` ev dizinine genişlemez; değişken gerekiyorsa `"$HOME/veri"` kullanın. `$(komut)` komut çıktısını bir değişkene almak içindir; bu biçim komutu gerçekten çalıştırır.

5.4 Komuttan .sh dosyasına geçiş

Bash'te `nano betikler/say_satir.sh` komutuyla dosyayı açın. Aşağıdaki içeriği yazın; Nano'da `Ctrl+O`, Enter ve `Ctrl+X` ile kaydedip çıkın. `nano` yoksa sonraki bölümdeki Ubuntu araç kurulumu kullanılabilir. Dosya düz metin ve LF satır sonlarıyla kaydedilmelidir.

LINUX / BASH

```
#!/usr/bin/env bash
set -euo pipefail

if [[ $# -ne 1 ]]; then
    printf 'Usage: bash say_satir.sh input.tsv\n' >&2
    exit 2
fi

girdi="$1"
if [[ ! -f "$girdi" ]]; then
    printf 'Input file not found: %s\n' "$girdi" >&2
    exit 1
fi
wc -l < "$girdi"
```

İlk satırdaki **shebang**, doğrudan çalıştırmada Bash'in bulunmasını ister. `$#` argüman sayısı, `$1` ilk argümandır. `[[ ... ]]` koşulu sınar; `-f` normal bir dosyanın varlığını denetler. `if` ile başlayan bölüm `fi` ile kapanır. Argüman sayısını önce kontrol etmek, `set -u` açıkken tanımsız `$1` okumayı önler.

`set -e`, bazı başarısız komutlarda betiği durdurur; koşullar ve bazı komut birleşimleri gibi istisnaları vardır. `-u` tanımsız değişken kullanımını, `pipefail` ise boru içindeki hataları görünür kılar. Bu üç seçenek bilimsel doğrulama yapmaz ve açık girdi kontrollerinin yerini tutmaz.[19]

LINUX / BASH

```
bash -n betikler/say_satir.sh
bash betikler/say_satir.sh veri/uzunluk.tsv
chmod u+x betikler/say_satir.sh
./betikler/say_satir.sh veri/uzunluk.tsv
```

`bash -n` sözdizimini inceler, analizi çalıştırmaz. Son iki çalıştırma yolu burada aynı sonucu, 4'ü üretir. `bash dosya.sh` için çalıştırma izni gerekmez; `./dosya.sh` doğrudan başlatmadır ve izin gerektirir. `sh dosya.sh` yazmak Bash'e özgü sözdizimini yanlış kabukla yorumlatabilir.

5.5 Döngü ve okunabilirlik

Döngü, benzer işlemi birden çok girdi üzerinde tekrarlamak içindir. Aşağıdaki örnek her TSV dosyasının adını ve satır sayısını ayrı verir. `[[ -f ... ]]` kontrolü, eşleşen dosya olmadığında genişlemeyen `*.tsv` metnini dosya sanmayı önler.

LINUX / BASH

```
for girdi in veri/*.tsv; do
    [[ -f "$girdi" ]] || continue
    printf '%s: ' "$girdi"
    wc -l < "$girdi"
done
```

Bir satırda birçok işi sıkıştırmak yerine anlamlı değişken adları, tutarlı girinti ve açık girdi/çıktı yolları kullanın. Uzun komutlarda satır sonundaki \ devam işaretidir; ardından boşluk koymayın. Betik bir insan tarafından okunabilir olmalı; daha kısa olması tek başına daha iyi olduğu anlamına gelmez.

Alıştırma. Betiği argümansız ve var olmayan bir dosyayla çağırın. Kullanım hatasında 2, dosya bulunamadığında 1 dönmesini bekleyin. Ardından gerçek dosyada 4 sonucunu doğrulayın. Başarılı bir örnek kadar beklenen bir başarısızlık da betiğin davranışını öğretir.

6 Conda ekosistemi ve proje ortamları

Bir komutun çalışması için programın kendisi kadar bağımlılıklarının da uyumlu olması gerekir. Bir analiz eski bir Python sürümü, başka bir analiz farklı bir kütüphane isteyebilir. **Ortam**, belirli bir iş için yazılımları ve bağımlılıklarını birlikte tutan dizindir. Ortam oluşturmak sanal makine oluşturmak değildir; ortam, işletim sistemi çekirdeğini veya CPU mimarisini değiştirmez.

6.1 Hangi ad hangi rolü anlatır?

Conda, paket ve ortam yöneticisidir; yalnızca Python paketlerini değil başka dillerdeki araçları da kurabilir. **Miniconda**, Conda'yı ve temel bileşenleri getiren küçük bir kurucudur. **Miniforge**, Conda ve Mamba'yı conda-forge merkezli bir başlangıç yapılandırmasıyla sunan ayrı bir dağıtımdır. İkisini birden kurmak zorunlu değildir.[20, 21]

Mamba, Conda paketleriyle ortam yöneten ayrı bir komut satırı aracıdır. **libmamba** ise paket çözümleme işinde kullanılan kütüphanedir. Conda, `conda-libmamba-solver` eklentisi üzerinden bunu kullanır; resmî belgeler, libmamba'nın Conda 23.10'dan itibaren varsayılan çözücü olduğunu belirtir. Bu nedenle güncel Conda ile Mamba'yı yalnızca eski Conda çözücüsünün hızı üzerinden karşılaştırmayın. İndirme, paket açma ve diske yazma da kurulum süresinin parçalarıdır.[22, 23]

6.2 Miniconda'yı Linux veya WSL içine kurma

WSL kullanıyorsanız bu adımları **Ubuntu/Bash içinde** çalıştırın. Windows için indirilen `.exe` kurucusu, bu Linux ortamına kurulum yapmaz. Doğrudan Ubuntu kullanan öğrenci aynı Linux adımlarını izler. Önce Ubuntu'nun paket listesini ve küçük yardımcıları hazırlayın.[8]

LINUX / BASH

```
sudo apt update
sudo apt install curl ca-certificates nano less python3
uname -m
```

`apt update` kurulmuş bütün paketleri yükseltmez; depolardaki paket listesini yeniler. `apt install` sistem araçlarını kurar. `sudo` burada sistem paket yöneticisi için gereklidir; Miniconda'yı ev dizinine kurarken kullanılmaz. Sistem Python'unun dosyalarını Conda ile değiştirmeyeceğiz.

Aşağıdaki örnek `uname -m` sonucu `x86_64` olan Linux içindir. Sonuç `aarch64` ise resmî sayfadaki ARM64 kurucusunun donanım koşullarını kontrol edip dosya adında `x86_64` yerine `aarch64` kullanın. ARM paketinin bulunması, her biyoinformatik aracının o mimaride bulunduğunu göstermez.[20, 26]

LINUX / BASH

```
cd ~
mkdir miniconda-kurulum
cd miniconda-kurulum
curl -fLO https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
sha256sum Miniconda3-latest-Linux-x86_64.sh
```

İndirilen dosyanın özetini resmî Miniconda arşivindeki *aynı dosyaya* ait SHA-256 ile karşılaştırın. Eşleşmiyorsa çalıştırmayın. `latest` değişen bir hedeftir; indirilen sürümü ve kurucu dosyasını kendi kurulum kaydınızda saklayın. Kurumunuzun paket deposu kullanım koşullarını da kontrol edin. Doğrulamadan sonra kurucuyu başlatın.[20]

LINUX / BASH

```
bash Miniconda3-latest-Linux-x86_64.sh
```

Lisansı okuyun, ev dizininizdeki önerilen kurulum yerini değerlendirin ve Bash başlatma dosyasını düzenleyen `conda init` sorusuna uygun yanıt verin. Var olan Miniconda kurulumunun üzerine yeni kurucu çalıştırmayın. Kurulum bitince Ubuntu/Bash terminalini kapatıp yeniden açın. Başlatmayı kabul etmediyseniz kurulumun varsayılan yerde olduğunu doğruladıktan sonra şu yolu kullanabilirsiniz.

LINUX / BASH

```
~/miniconda3/bin/conda init bash
```

Bu işlemden sonra da yeni bir Bash oturumu açın. Kurulumun doğrulanması için sürüm, yapılandırma ve paket listesine bakın.

LINUX / BASH

```
conda --version
conda info
conda config --show solver
conda list
```

`conda info` Linux platformunu göstermelidir. Eski bir yapılandırma `classic` çözücüsünü seçmiş olabilir; varsayılanla gerçek ayarı karıştırmayın. Base ortamı paket yöneticisinin bakım alanıdır; dersin analiz paketlerini buraya yığmayın.

6.3 Miniconda üzerinden Mamba kullanma

Mamba'nın resmî yeni kurulum önerisi Miniforge'dur. Miniconda ile başlamış bir öğrenci için burada Mamba'yı, yalnızca conda-forge paketleri içeren ayrı bir yönetim ortamına kuruyoruz. Bu, Miniconda'nın base ortamına farklı kanal paketleri karıştırmadan Mamba komutunu öğrenmeye yarar. conda-forge da farklı kaynakları denemek için yalıtılmış ortam yolunu belgeler.[24, 25]

LINUX / BASH

```
conda create -n mamba-tools --override-channels -c conda-forge mamba
conda run -n mamba-tools mamba --version
```

`-n` ortamın adıdır. `--override-channels`, bu işlemde kullanıcının diğer kanallarını dışarıda bırakır. `conda run`, etkin ortamı değiştirmeden adı verilen ortamın içindeki komutu çalıştırır. Kurulum işleminin planını onaylamadan önce okuyun. `mamba-tools` zaten varsa onu silmek yerine içeriğini ve sürümünü kontrol edin.

Şimdi Mamba ile ders ortamını oluşturalım. `--no-capture-output`, kurulum planının ve onay isteğinin terminalde doğrudan görünmesini sağlar.[35] `-p`, ortamı açık bir dizine kurar; böylece yönetim ortamı ile ders ortamının yerleri karışmaz. Python 3.12 burada eğitim için seçilmiş sürüm ailesidir, "en yeni sürüm" iddiası değildir.

LINUX / BASH

```
conda run --no-capture-output -n mamba-tools mamba create \
    -p "$HOME/conda-envs/terminal-bio" \
    --override-channels -c conda-forge -c bioconda \
    --strict-channel-priority python=3.12 seqkit
conda activate "$HOME/conda-envs/terminal-bio"
python --version
command -v python
seqkit version
```

`command -v python` etkin ders ortamının içindeki yolu göstermelidir. `seqkit`, Bioconda'dan bir araç kurmayı göstermek içindir; bu notun TSV uygulaması yalnızca Python standart kütüphanesini kullanır. Araç belirli bir mimaride bulunmuyorsa başka mimariyi zorlayarak kurmayın; paketin desteklediği platformu kontrol edin.

Miniforge ile başlayan öğrencide `mamba` zaten bulunur; aynı oluşturma komutu doğrudan `mamba create` ile başlatılır; ayrı yönetim ortamı gerekmez. Bu iki yol alternatiftir. İki farklı kurulumun başlatma satırlarını aynı kabuğa gelişigüzel eklemek, hangi Conda'nın çalıştığını belirsizleştirir.

6.4 Kanallar, sürümler ve pip

Kanal, paketlerin indirildiği depodur. conda-forge genel bağımlılıkları, Bioconda biyoinformatik paketlerini sağlar. Gösterilen sırada conda-forge daha yüksek önceliklidir. `--strict-channel-priority`, aynı paket üst kanalda bulunduğunda alt kanaldaki adayları dışlar; yalnızca en yeni görünen sürümleri rastgele birleştirmekten kaçınır. Bioconda'nın güncel önerisinde `defaults` yoktur. conda-forge ile `defaults` paketlerini aynı ortamda karıştırmak desteklenen bir geçiş yolu değildir.[26, 25]

Sürümü görmek için `conda list`; etkin ortamdan çıkmak için `conda deactivate` kullanılır. Paket değişikliklerini bitmiş analizin ortamında yapmak yerine yeni bir deneme ortamında sınayın. Aşağıdaki ilk komut yalnızca önerilen değişiklik planını gösterir, ikincisi gerçekten uygular; ikisini de Linux/Bash'te, ders ortamı etkinken kullanın.

LINUX / BASH

```
conda install --dry-run --override-channels \
    -c conda-forge -c bioconda --strict-channel-priority python=3.12
conda install --override-channels \
    -c conda-forge -c bioconda --strict-channel-priority python=3.12
```

`python=3.12`, 3.12 ailesini sınırlar; tam yama sürümünü ve bağımlılık yapılarını sabitlemez. Paket çözümleyici bu isteğe uygun bir bileşim seçer. `--dry-run` başarılı olsa bile paket henüz kurulmuş veya gerçek analizde denenmiş değildir.

pip, Python paket yöneticisidir; Conda'nın bütün paketlerini yönetmez. Gerekliyse önce Conda bağımlılıklarını kurun, sonra etkin ortamın Python'u ile `python -m pip` kullanın. `sudo pip` veya `pip --user` ile bu proje ortamının dışına paket taşımayın. Pip ile eklenen paketler Conda'nın çözümlemesiyle her zaman tam uyumlu izlenmez; sonraki değişikliklerde ortamı tanımdan yeniden oluşturmak daha anlaşılır olabilir. Bu dersin uygulamasında pip paketi gerekmez.[27]

6.5 Ortamı yeniden oluşturma

Bir ortam adı tek başına yeniden üretilebilirlik kaydı değildir. Proje dizininde, daha önce aynı adlı kayıtlar yokken aşağıdaki dosyaları üretin. `>` var olan dosyanın üzerine yazabilir.

LINUX / BASH

```
conda env export --from-history > environment-history.yml
conda env export > environment-full.yml
conda list --explicit > environment-explicit.txt
```

Geçmişten dışa aktarma, istediğiniz üst düzey paketleri kaydeder; daha taşınabilirdir ama aynı paket bileşimini garanti etmez. Tam YAML, çözülen bağımlılıkları da içerir; platforma bağlı yapı ve yerel `prefix` alanı bulunabilir. Açık paket listesi, aynı platformdaki somut paket yapılarını yeniden kurmak içindir; pip ile kurulmuş paketleri kapsayan genel bir kilit değildir. Paylaşmadan önce yerel yolları ve özel kanal bilgilerini inceleyin.[27, 23]

LINUX / BASH

```
conda env create -n terminal-tekrar -f environment-history.yml
conda create -n terminal-ayni-platform --file environment-explicit.txt
```

Bunlar iki farklı yeniden oluşturma yoludur; ders sırasında ikisini birden kurmanız gerekmez. Aynı `linux-64` paket listesi uyumlu x86_64 Linux ve WSL2 için kullanılabilir; ARM64 veya Windows ortamı için aynı listeyi zorlamayın. Kurulumdan sonra sürümleri ve küçük uygulamanın çıktısını tekrar doğrulayın. Paketlerin bulunması, işletim sistemi kitaplıklarının veya sonuçların otomatik olarak eşdeğer olduğu anlamına gelmez.

7 Python'a temel giriş

Bash dosyaları ve programları bir araya getirmekte kullanışlıdır. Bir tablonun her satırından yeni bir nicelik hesaplamak istediğimizde Python, verinin yapısını açıkça ifade etmeyi kolaylaştırır. Bu bölümde amaç ileri programlama değil; küçük bir metin dosyasını okuyup hesaplayarak yeni bir tablo üretmektir.

7.1 Yorumlayıcı ve betik

Python yorumlayıcısı, Python kodunu çalıştıran programdır. Linux/Bash'te etkin ders ortamında aşağıdaki komutla etkileşimli yorumlayıcıyı açın.

LINUX / BASH

```
python
```

`>>>` istemi göründüğünde Python kodu yazabilirsiniz; `>>>` işaretini kendiniz yazmayın. Aşağıdaki satırları yorumlayıcıda sırayla deneyin.

PYTHON

```
2 + 3
len("ACGT")
exit()
```

İlk iki yanıt 5 ve 4'tür. `exit()` Bash'e döndürür. Buna karşılık `.py` dosyası kodu saklar; `python dosya.py` komutu bu dosyayı çalıştırır. Betikte bir değerin görünmesi için genellikle `print` gerekir; etkileşimli yorumlayıcının son ifadenin değerini göstermesiyle bunu karıştırmayın.[28]

7.2 Değerler, adlar ve veri yapıları

Bir değişken adı bir değere bağlanır. Metin `str`, tam sayı `int`, ondalıklı sayı `float`, doğru/yanlış değeri `bool` türündedir. Python kodunda ondalık ayıracı noktadır. Türkçe metindeki sayı yazımını koda taşımayın.

PYTHON

```
sequence = "ACGTACGT"
length = len(sequence)
gc_count = sequence.count("G") + sequence.count("C")
gc_percent = 100 * gc_count / length
is_long = length >= 8
print(length, gc_count, gc_percent, is_long)
```

Çıktı `8 4 50.0 True` olur. `=` atama, `==` eşitlik sınamasıdır. Bir harf sayısını yüzdeye çevirirken paydanın ne olduğunu açıkça belirlemek gerekir; birazdan belirsiz bazları ayrı ele alacağız.

Liste, sıralı değerler tutar; **sözlük**, bir anahtarı bir değerle ilişkilendirir. Dizinin kendisiyle kaydın adını ayırmak için sözlük kullanabiliriz. Python'da indeksleme sıfırdan başlar.

PYTHON

```
lengths = [8, 12, 8]
record = {"record_id": "seq_a", "sequence": "ACGTACGT"}
print(lengths[0])
print(record["record_id"])
print(record["sequence"][:4])
```

Sırasıyla 8, `seq_a` ve `ACGT` yazılır. `[:4]` ilk dört karakteri alır; bitiş indeksi dahil değildir. Sözlükte kayıt kimliği bulunmayan bir anahtarı istemek hata verir. Hata, çoğu zaman veri hakkında yanlış bir varsayımı ortaya çıkarır.

7.3 Girinti, koşul, döngü ve fonksiyon

Python'da girinti kodun yapısını belirler; süs değildir. Koşula veya döngüye ait satırları dört boşlukla girintileyin ve sekmeyle boşluğu karıştırmayın. Aşağıdaki kodu bir `.py` dosyasına yazın veya yorumlayıcıya blok olarak girin. Etkileşimli kullanımda bloğun bittiğini boş satırla bildirirsiniz.[29]

PYTHON

```
lengths = [8, 12, 8]
for length in lengths:
    if length >= 10:
        print(length, "keep")
    else:
        print(length, "short")
```

Çıktı sırasıyla `8 short`, `12 keep`, `8 short` olur. Eşiği 10'dan 8'e indirirseniz üç kayıt da tutulur. Bu eşik burada programlama örneğidir; biyolojik kalite ölçütü olarak önerilmez.

Fonksiyon, adı olan ve yeniden çağrılabilen bir işlemdir. Aşağıdaki fonksiyon bir dizi alır ve uzunluğunu döndürür. `return`, sonucu çağıran koda verir; `print` ise ekrana yazar.

PYTHON

```
def sequence_length(sequence):
    return len(sequence)

result = sequence_length("ACGT")
print(result)
```

Sonuç 4'tür. Fonksiyon yazmak için her küçük satırı ayrı bir yapıya dönüştürmek gerekmez; yinelenen veya adı verilince daha anlaşılır olan işlemler iyi adaylardır.

7.4 Dosya okuma ve yazma

Bash'te `~/terminal-pratik` dizinindeyken `nano betikler/oku.py` ile bir dosya oluşturup aşağıdaki Python kodunu kaydedin. `with` bloğu bitince dosyayı kapatır. `encoding="utf-8"`, metnin karakter kodlamasını açıkça seçer.[30]

PYTHON

```
with open("veri/uzunluk.tsv", encoding="utf-8") as handle:
    for line in handle:
        print(line.rstrip("\n"))
```

Bash'e dönüp çalıştırın.

LINUX / BASH

```
python betikler/oku.py
```

Betik bulunduğu dizine otomatik geçmez. Göreli girdi yolu, Python'u başlattığınız çalışma dizinine göre yorumlanır. Aynı kodu `betikler` içinden çalıştırırsanız `veri` dizini bulunmayabilir. Bu yüzden uygulamaların çalışma dizinini belgelemek gerekir.

Yeni bir dosyaya yazmak için `open` kipini seçeriz. `"w"` var olan içeriği sıfırlar; `"a"` sona ekler; `"x"` yalnızca yeni dosya oluşturur ve ad zaten varsa hata verir. Eğitim uygulamasında sonuçların yanlışlıkla ezilmemesi için `"x"` kullanacağız.

PYTHON

```
with open("sonuc/mesaj.txt", "x", encoding="utf-8") as handle:
    handle.write("Three synthetic records\n")
```

Bu kodu ikinci kez çalıştırınca dosyanın zaten var olduğuna ilişkin hata beklenir. Yeniden çalıştırma, önceki sonucu sessizce silmek zorunda değildir; yeni bir çıktı adı seçebilirsiniz.

7.5 Tabloyu satır satır anlamlandırma

`csv.DictReader`, her satırı sütun adlarıyla erişilen bir sözlüğe dönüştürür. TSV için ayraç `"\t"` olmalıdır; varsayılan virgül ayıracını kullanmak sütunları yanlış okur. Dosyadan gelen sayılar başlangıçta metindir; aritmetik için dönüştürülür.[14]

PYTHON

```
import csv

with open("veri/uzunluk.tsv", encoding="utf-8", newline="") as handle:
    reader = csv.DictReader(handle, delimiter="\t")
    for row in reader:
        length = int(row["length"])
        if length >= 10:
            print(row["id"], length)
```

Beklenen çıktı `seq_b 12`'dir. `import`, standart kütüphanedeki modülü kullanılabilir kılar; `csv` için pip kurulumu gerekmez. `int("sekiz")` geçerli bir dönüşüm değildir. Böyle bir hatayı sessizce sıfıra çevirmek, bozuk girdiyi geçerli veri gibi gösterebilir.

Alıştırma. Aynı tabloyu, eşiği 8 yaparak okuyun; üç kayıt bekleyin. Sonra tablonun bir kopyasında bir uzunluğu `?` yapın. Dönüşümün başarısız olmasını ve yanlış bir sayısal sonuç üretilmemesini bekleyin. Asıl eğitim girdisini değiştirmeden kopya üzerinde çalışın.

8 Performans ve I/O darboğazları

Bir analiz yavaşladığında ilk soru "kaç çekirdek kullanıyor?" ile sınırlı olmamalıdır. Program hesaplama yapıyor, diskten veri bekliyor, ağdan indiriyor veya belleğe sığmaya çalışıyor olabilir. **Darboğaz**, o aşamanın hızını sınırlayan kaynaktır; bir iş akışının farklı adımlarında değişebilir.

8.1 CPU, RAM, disk ve ağı ayırma

CPU hesaplama yapar. RAM, aktif verileri ve program durumunu tutar. Disk kalıcı depolamadır; ağ uzak kaynağa erişim sağlar. Bir referans dosyasının indirilmesi ağla, sıkıştırılmış dosyanın açılması CPU ve diskle, büyük bir indeksin yüklenmesi disk ve RAM ile sınırlanabilir. Bunlar birbirini dışlayan sınıflar değildir.

Çok sayıda küçük dosya, toplam boyut küçük olsa bile ad bulma, açma, kapatma ve üstveri işlemlerini artırır. Büyük bir dosyayı baştan sona **ardışık okumak** ile farklı konumlardan küçük parçalar hâlinde **rastgele okumak** aynı erişim örüntüsü değildir. İşletim sistemi yakın zamanda okunan veriyi RAM önbelleğinde tutabilir; ikinci çalışmanın hızlanması, programın değiştiğini göstermez.

Sıkıştırma diskte ve ağda taşınan baytları azaltırken açma işlemi için CPU harcar. Bu nedenle sıkıştırılmış girdi her zaman daha yavaş veya daha hızlı değildir. Swap'a yoğun başvurulan bir işte daha fazla CPU vermek sorunu çözmeyebilir. Linux'taki bellek ve süreç özetini şu komutlarla inceleyin. `top` ekranından `q` ile çıkılır; `vmstat 1 5` beş gözlem yazar.

LINUX / BASH

```
top
free -h
df -h .
du -sh veri
vmstat 1 5
```

`df` dosya sisteminin kapasitesini, `du` belirtilen dizinin kapladığı alanı özetler. `vmstat` çıktısındaki `si`/`so`, swap giriş/çıkış etkinliğini izlemeye yardımcı olur; ilk satır açılıştan beri ortalamalar içerdiğinden sonraki aralıklarla aynı şekilde okunmaz. Tek bir görüntüden kesin darboğaz tanısı koymayın. WSL'de Windows Görev Yöneticisi'ni de birlikte inceleyin.[31, 12]

8.2 Üç çalışma yolunu ayrı düşünme

Doğrudan Linux kurulumunda uygulama yerel Linux çekirdeği ve bağlı dosya sistemi üzerinden çalışır. Dosya sistemi ext4 olmak zorunda değildir; ağ diski de kullanılabilir. **WSL2'nin Linux dizininde** uygulama Linux çekirdeği içinde, çoğunlukla ext4 sanal disk üzerinden çalışır. **WSL'den Windows dosyalarına erişimde** ise `/mnt/c` gibi bir bağlama yoluyla işletim sistemleri arası dosya paylaşım katmanı devreye girer.[4, 11]

Bunlar tek bir "Linux ve Windows" karşılaştırmasına indirgenemez. Özellikle küçük dosya ve üstveri yoğun işlemlerde yol seçimi önemlidir. Windows paylaşımının uygulanışı WSL sürümü ve ayarlarıyla değişebilir; güncel belgelerdeki deneysel dosya paylaşımı seçeneklerini eski ölçümlere geriye dönük uygulamayın.[10] Ders için ortak başlangıç, Linux araçlarını Linux ev dizinindeki projede çalıştırmak ve Windows'a gereken son çıktıları kopyalamaktır.

8.3 Birincil çalışmalardan ne çıkarılabilir?

Aşağıdaki çalışmalar aynı soruyu aynı koşullarda sınamaz. Donanım, yazılım ve erişim yolu bilgileri birlikte okunmalıdır; bildirilmeyen bir ayrıntıyı varsayarak tamamlamıyoruz.

Doğrudan WSL2 karşılaştırması. Anshori ve arkadaşları (2026), Acer Aspire Lite 14 üzerinde Intel i3-1215U ve 8 GB DDR5 ile CPU tabanlı dil modeli çıkarımını karşılaştırır. Windows 11 Home, WSL 2.6.3.0/çekirdek 6.6.87.2-1, hem yerel hem konuk tarafta Ubuntu Server 24.04.3 LTS ve Ollama 0.24.0 bildirilir. Llama3.2:3B, Qwen2.5:3B ve Phi3.5:3.8B için iki ısınma ve on ölçüm kullanılır. Modeller önceden belleğe yüklenir. Örneğin hafif istemde Llama3.2 için ortalama gecikme WSL2'de 12,73 s, yerel Linux'ta 10,83 s olarak verilir. Bu, dizi analizi veya dosya okuma ölçümü değildir. Makale dosya sistemi türünü, model dosyasının Linux/Windows tarafındaki yerini ve WSL bellek üst sınırını belirtmez; dosya sistemi etkisini buradan ayıramayız.[32]

Bilimsel hesaplama için teknik ölçüm. Kinghorn'un (2022) hakemli makale olmayan birincil teknik raporu, aynı Ryzen 5800X/64 GB DDR4 bilgisayarda Ubuntu 22.04 ile Windows 11 üzerindeki WSL2/Ubuntu 22.04'ü karşılaştırır. WSL belleği 56 GB'a çıkarılmıştır. HPL, HPCG ve NAMD; Zen3 için derlenmiş, Docker'da hazırlanıp enroot ile çalıştırılan paketlerdir. HPL'de fark belirginleşirken HPCG ve NAMD sonuçları yakın bulunur. AOCC 3.2 ve AMDFFTW 3.2 belirtilir; Windows build'i, WSL çekirdek sürümü, uygulamaların tüm sürümleri ve dosya sistemi/çalışma yolu verilmez. Bu rapor iş yüküne bağlılığı gösterir; konteyner içindeki hesaplama sonucunu doğrudan bütün Bash veya dosya işlemlerine taşımayız.[33]

Docker katmanını ayırma. Sobieraj ve Kotyński (2024), Intel i5-8257U, 16 GB RAM ve 256 GB NVMe içeren MacBook Pro'da Windows 10 22H2, T2 uyarlamalı Ubuntu 22.04/6.4.8 ve macOS 13.5.1 üzerinde Docker iş yüklerini karşılaştırır. Makalede Docker sürümü "4.20" olarak adlandırılır; Engine/Desktop ayrımı bu kayıtta net değildir. Hesaplama, Sysbench, dosya ve ağ testleri yapılır; Windows'taki yol Docker Desktop ve WSL2'yi birlikte içerir. WSL çekirdeği ile ana sistem dosya sistemi ve bağlama konumu yeterince tanımlanmaz. Dolayısıyla sonuçlar Docker dağıtımlarına ilişkindir; konteynersiz WSL2 için bir yavaşlama katsayısı olarak kullanılamaz.[34]

Bu sınırlı ve farklı iş yüklerine ait kanıt, WSL2 için evrensel bir yavaşlama yüzdesini desteklemez. WSL1 ölçümleri de ayrı mimariye aittir. Bu notta yerel Linux–WSL2 performansı ölçülmemiştir; sunulan sayılar ilgili çalışmanın bildirdiği değerlerdir. Bilgisayarınızda beklenen süreyi tahmin etmek için kendi iş yükünüze yakın, küçük ama temsil edici bir deneme gerekir.

8.4 Adil bir küçük karşılaştırma tasarlama

Önce çıktının doğru ve aynı olduğunu doğrulayın; yanlış sonuç daha hızlı üretildi diye yöntem üstün sayılmaz. Aynı makineyi, girdiyi, araç sürümünü, parametreleri ve iş parçacığı sayısını kullanın. Windows build'i, WSL sürümü, dağıtım, çekirdek, bellek/swap sınırı ve dosyanın gerçek bağlama yerini kaydedin. Linux tarafında ilgili yolu incelemek için aşağıdaki komutlar kullanılabilir.

LINUX / BASH

```
findmnt -T .
uname -r
uname -m
```

`findmnt` bağlama bilgisini gösterir; sanal diskin Windows üzerindeki fiziksel yerleşimini bütünüyle açıklamaz. WSL2 içindeki iki dizini karşılaştırmak dosya yolu etkisini inceleyebilir; tek başına doğrudan Linux ile karşılaştırma değildir.

Bölüm 9'teki uygulama hazır olduğunda süre ölçme sözdizimi aşağıdaki gibi denenebilir. Bu bir yöntem örneğidir; önceden ölçülmüş süre verilmez. Çıktı adı yeni olmalıdır.

LINUX / BASH

```
time python scripts/02_sequence_summary.py \
    work/input.tsv work/timed-summary.tsv 8
```

`real` geçen duvar süresini, `user` kullanıcı kodundaki, `sys` çekirdekteki CPU süresini özetler. Çok iş parçacıklı programlarda CPU süreleri duvar süresini aşabilir. Bu üç kayıt o kadar küçüktür ki yorumlayıcı başlatma gideri baskın olabilir; ciddi bir performans sıralaması için uygun değildir. Daha büyük denemede ilk çalıştırmayı sonraki önbellekli çalıştırmalardan ayırın, birkaç tekrar ve dağılım raporlayın; arka plan yükünü ve güç durumunu sabit tutmaya çalışın. Karşılaştırma uğruna sistem genelindeki önbellekleri rastgele temizlemeyin.

Alıştırma. `/mnt/c` altındaki bir projede CPU kullanımı düşükken dosya açma işlemleri uzun sürüyor. "Daha çok çekirdek" sonucuna varmadan hangi denemeyi yaparsınız? Aynı küçük girdi ve aynı komutun Linux ev dizinindeki kopyasını deneyip çıktı eşitliğini, süreyi ve disk etkinliğini karşılaştırırsınız. Fark bulmak erişim yoluna ilişkin kanıttır; tek başına bütün nedenleri ayırmaz.

9 Bütünleştirici uygulama ve tekrarlanabilirlik

Şimdi dosya yönetimini, Bash'i ve Python'u tek bir küçük soruda birleştireceğiz: Üç sentetik dizinin uzunluğu ve bilinen bazları içindeki GC yüzdesi nedir; seçilen uzunluk eşiği hangi kayıtları tutar? Bu hesaplama bir kalite değerlendirmesi veya tür tayini değildir. Amaç, açık bir tanımdan denetlenebilir bir sonuca gitmektir.

9.1 Çalışma dizini ve başlangıç girdisi

Notla birlikte verilen `main_scripts/terminal_basics/` dizinini Linux ev dizininize `terminal-proje` adıyla kopyalayın. Aşağıdaki komutlar, notlar deposunun kökündeyken ve hedef henüz yokken kullanılabilir. Hedef zaten varsa içeriğini koruyup yeni bir ad seçin.

LINUX / BASH

```
cp -R main_scripts/terminal_basics "$HOME/terminal-proje"
cd "$HOME/terminal-proje"
ls scripts
conda activate "$HOME/conda-envs/terminal-bio"
```

Son komut Bölüm 6'daki ortamı seçer. Depoyu Windows'a indirmiş bir öğrenci kopyalamayı WSL içinden yapmalı ve son çalışma dizinini Linux ev dizininde tutmalıdır. Yalnızca PDF'yi indirdiyseniz eşlik eden betik dizinini de edinin; aşağıdaki dosya adları bu dizinin içeriğidir.

Birinci aşama `work` adlı *yeni* bir dizin ve sentetik girdi oluşturur. Her aşamayı ayrı çalıştırıp sonucunu inceleyin.

LINUX / BASH

```
bash scripts/01_prepare.sh work
cat work/input.tsv
```

BEKLENEN ÇIKTI / DOSYA İÇERİĞİ

```
record_id   sequence
seq_a       ACGTACGT
seq_b       GGGCCCAT
seq_c       ATATATNN
```

Gerçek dosyanın ayıracı sekmedir. `01_prepare.sh`, mevcut bir çalışma dizininin üstüne yazmaz. Bu girdi FASTA değildir; iki sütunlu eğitim tablosudur. FASTA'da başlıklar `>` ile başlar ve bir dizi birden çok satıra bölünebilir; bir FASTA dosyasını bu betiğe doğrudan vermeyin.

9.2 Niceliğin tanımı ve parametrenin etkisi

Dizi uzunluğu `N` dahil bütün karakterlerin sayısıdır. `called_bases` yalnızca A, C, G ve T sayısını içerir. GC yüzdesini bu bilinen bazlar üzerinden tanımlıyoruz:

$$\text{GC yüzdesi} = 100 \times \frac{n_G + n_C}{n_A + n_C + n_G + n_T}.$$

Bu eğitim tanımı, belirsiz bazları paydadan çıkarır. Başka bir araç toplam uzunluğu payda yapabilir; aynı isimli sütunları karşılaştırmadan tanımı okuyun. Tamamı `N` olan bir dizide payda sıfırdır; yüzdeyi 0 yerine `NA` olarak yazarız. Sıfır GC ile bilinmeyen GC aynı bilgi değildir.

`min_length`, kaydı tutmak için gereken toplam uzunluğu belirler. Burada 8 seçilmesinin nedeni üç küçük diziyi birlikte incelemektir; biyolojik bir eşik değildir. `keep=yes`, yalnızca uzunluk koşulunun sağlandığını gösterir. Belirsiz baz içeren bir kayıt da bu koşulu sağlayabilir; bu sütun kalite onayı olarak yorumlanmaz.

İkinci aşamanın temel hesabı aşağıdaki fonksiyondur. Tam betik ayrıca başlıkları, kimlik tekilliğini, satır yapısını, geçerli alfabeyi ve çıktı çakışmasını denetler. Küçük harfler veya başka IUPAC belirsizlik kodları bu eğitim girdisinin kapsamı dışındadır; sessizce düzeltilmez.

PYTHON

```
def sequence_stats(sequence):
    if not sequence or any(base not in "ACGTN" for base in sequence):
        raise ValueError("Expected a nonempty uppercase A/C/G/T/N sequence")
    called = sum(sequence.count(base) for base in "ACGT")
    gc = sequence.count("G") + sequence.count("C")
    percent = 100 * gc / called if called else None
    return len(sequence), called, percent
```

`sum(... for ...)` her baz için sayımı toplayan kısa bir döngüdür. `any(...)` izin verilmeyen en az bir karakter olup olmadığını sınar. `None`, hesaplanabilir bir yüzde olmadığını temsil eder; dosyaya yazılırken `NA`'ya

dönüştürülür. Fonksiyonun üç dönüş değeri, uzunluk ile baz sayısını birbirine karıştırmadan kullanmayı sağlar.

9.3 Hesabı çalıştırma ve sonucu doğrulama

Linux/Bash'te proje dizininden ikinci aşamayı çalıştırın. Girdi yolu, çıktı yolu ve eşik üç ayrı argümandır.

LINUX / BASH

```
python scripts/02_sequence_summary.py work/input.tsv work/summary.tsv 8
cat work/summary.tsv
python scripts/03_check_example.py work/summary.tsv
```

İkinci aşama `Wrote 3 records`; denetleyici ise üç sentetik kaydın elle hesapla uyuştuğunu belirten `OK` iletisi üretmelidir. Beklenen tablo şöyledir:

BEKLENEN ÇIKTI / DOSYA İÇERİĞİ

```
record_id  length  called_bases  gc_percent  keep
seq_a      8       8             50.000      yes
seq_b      8       8             75.000      yes
seq_c      8       6              0.000      yes
```

`seq_a` için dört GC bazı/sekiz bilinen baz yüzde 50'dir. `seq_b` için altı/sekiz yüzde 75; `seq_c` için sıfır/altı yüzde 0'dır. Üç kaydın toplam uzunluğu 24, bilinen baz sayısı 22 ve GC sayısı 10'dur. Toplu oran istenirse $100 \times 10/22 \simeq 45{,}45$ hesaplanır. Satır yüzdelerinin basit ortalaması 41,67'dir; paydalar farklı olduğu için bu iki nicelik aynı değildir. Betik burada yalnızca satır düzeyindeki sonuçları verir.

Denetleyici genel amaçlı bir dizi doğrulayıcı değildir; bu üç kayıt ve `min_length=8` için elle hesaplanmış beklentiyi sınar. Sütunları, kayıt kimliğini/sırasını, sayıları ve sonlu GC değerlerini kontrol eder. Eşiği veya girdiyi değiştirince aynı beklenen tabloyu kullanmak doğru değildir.

9.4 Açıklamalı deneyler ve negatif kontroller

Eşiği 9'a çıkarıp yeni çıktı adına yazın. Üç satır korunur ama `keep` hepsinde `no` olur; uzunluklar ve GC yüzdeleri değişmez. Filtreleme kararını sonuçtan ayırarak görmek, parametre etkisini anlaşılır kılar.

LINUX / BASH

```
python scripts/02_sequence_summary.py work/input.tsv work/summary-min9.tsv 9
cat work/summary-min9.tsv
```

Bu dosyayı `03_check_example.py`'ye verirseniz hata beklenir; denetleyicinin sabit eğitim beklentisi 8 eşiğine aittir. Hatanın nedeni biyolojik hesabın değişmesi değil, tutma kararının değişmesidir.

Diğer alıştırmaları girdinin kopyalarında yapın. `N` içeren diziye bir `G` eklemek hem uzunluğu hem bilinen baz sayısını artırır; yeni GC paydasını elle hesaplayın. Bir diziye `X` eklemek, yinelenen kimlik oluşturmak veya bir alanı eksiltmek betiği durdurmalıdır. Tamamı `N` olan, uzunluğu 8 bir kayıt için `called_bases=0`, `gc_percent=NA` ve `keep=yes` beklenir. Bu örnek, uzunluk filtresinin belirsiz baz sorununu çözmediğini gösterir.

Sonuç dosyasının bir kopyasında `50.000` değerini `51.000` yapıp denetleyiciyi çalıştırın. Sıfırdan farklı çıkış ve GC uyumsuzluğu iletisi bekleyin. `NaN` gibi sonlu olmayan bir değer de geçmemelidir. Yalnızca doğru girdide başarı veren bir kontrolün hatayı yakaladığını henüz göstermiş olmayız; bu nedenle negatif örnekler gereklidir.

9.5 Tekrarlanabilirlik kaydı

Yeniden üretilebilir çalışma; kod, girdi, parametre ve ortam bilgisinin birlikte korunmasını gerektirir. Komut geçmişi yararlı olsa da tek başına yeterli kayıt değildir. Başka dizinden çalıştırılan, farklı ortamda bulunan veya sonradan değiştirilen dosyaya erişen aynı komut farklı davranabilir.

Proje için hangi betiği hangi çalışma dizininden çağırdığınızı, girdi sürümünü, seçilen eşiği, beklenen çıktıyı ve değişiklik gerekçesini kısa bir çalışma notuna yazın. Aşağıdaki Linux/Bash komutları yeni bir ortam kaydı oluşturur; aynı adlı dosya varsa yeni ad seçin.

LINUX / BASH

```
{
    uname -srm
    bash --version
    python --version
    conda --version
    conda list
} > work/environment.txt 2>&1
sha256sum work/input.tsv work/summary.tsv > work/checksums.txt
```

`2>&1`, hata akışını o anda standart çıktının gittiği yere bağlar; burada sürüm iletilerini tek kayıtta tutmak içindir. Veri çıktısını böyle bir tanı günlüğüyle karıştırmayın. Özet değerler dosyanın sonradan değişip değişmediğini izler; biyolojik doğruluğu kanıtlamaz. Çalışma kaydını ve makineye özgü yolları yerelde saklayın; paylaşılacak belgeleri ayrıca gözden geçirin.

WSL kullananlar PowerShell'de `wsl --version` ve `wsl --list --verbose` çıktılarını da kaydetmelidir. Seçilen bellek, işlemci ve swap sınırları bu kayda eklenmelidir. Ortamı yeniden oluşturmak için Bölüm 6'daki dışa aktarma yollarından uygun olanı kullanın. Bu uygulamada rastgelelik yoktur; tohum seçmek gerekmemektedir. Paket sürümü kaydı, dosya özeti ve sonuç denetimi birbirini tamamlar.

9.6 Sık karşılaşılan sorunları sırayla ayırma

Komut bulunamıyor. Önce doğru kabukta olduğunuzu ve ortamın etkin olduğunu kontrol edin. Linux/Bash'te `command -v python` ve `conda info --envs`, yanlış Python seçimini ortaya çıkarabilir. Windows Python yolunu Linux ortamına eklemek çözüm değildir.

Dosya bulunamıyor. `pwd` ve `ls` ile çalışma dizinini, dosya adının harflerini ve yolun tırnaklanmasını kontrol edin. WSL'de Windows yolu `C:\...` biçimini Linux komutuna doğrudan taşımayın; Linux erişim yolunu kullanın.

İzin veya satır sonu hatası. Betiği `bash dosya.sh` ile çalıştırmak çalıştırma izni gerektirmez; `./dosya.sh` gerektirir. `bad interpreter` veya görünür `\r` hataları, Windows CRLF satır sonlarından kaynaklanabilir. Editörde LF seçip yalnızca ilgili metin dosyasını yeniden kaydedin; bütün veri dosyalarına körlemesine dönüşüm uygulamayın.

Çıktı zaten var. Eğitim betiğinin dosyayı ezmeyi reddetmesi tasarlanmış bir davranıştır. Yeni çıktı adı seçin; hangi çalışmanın hangi parametreye ait olduğunu kaydedin. Bir disk/yazma hatasından sonra kısmi dosya oluşmuş olabilir; varlığı veya sıfır çıkış kodu dışındaki içerik kontrollerini atlamayın.

Süreç sonlandırılıyor veya çok yavaşlıyor. Linux belleğini, WSL sınırını, Windows'taki yükü ve boş disk alanını birlikte inceleyin. `Killed` iletisi tek başına kesin neden söylemez; bellek yetersizliği olasılığı günlüklerle doğrulanmalıdır. Önce daha küçük girdi ve daha az eş zamanlı iş kullanın; sorunu yalnızca swap artırarak gizlemeyin.

Uygulama notu

Bu notun sentetik Bash/Python uygulaması ve negatif kontrolleri macOS arm64 üzerinde yerel olarak çalıştırılarak denetlenmiştir; kullanılan gerçek sürümler eşlik eden README'de kayıtlıdır. Linux/WSL2 hedefindeki kurulum, WSL Settings, .wslconfig ve paket çözümleme adımları resmî belgelere dayanan yönergelerdir; bu bilgisayarda çalıştırılmış Windows/WSL2 testleri olarak sunulmaz. Küçük örneğin başarısı büyük veri performansı veya bütün biyoinformatik araçlarının platform uyumluluğunu kanıtlamaz.

Kaynaklar

[1] Free Software Foundation. *Bash Reference Manual*. `https://www.gnu.org/software/bash/manual/bash.html`

[2] Microsoft. *What is PowerShell?* `https://learn.microsoft.com/en-us/powershell/scripting/overview`

[3] Microsoft. *What is Windows Terminal?* `https://learn.microsoft.com/en-us/windows/terminal/`

[4] Microsoft. *Comparing WSL Versions*. `https://learn.microsoft.com/en-us/windows/wsl/compare-versions`

[5] Canonical. *Install Ubuntu Desktop*. Ubuntu Desktop documentation. `https://ubuntu.com/desktop/docs/en/latest/tutorial/install-ubuntu-desktop/`

[6] Microsoft. *Install WSL*. `https://learn.microsoft.com/en-us/windows/wsl/install`

[7] Canonical. *Install Ubuntu on WSL 2*. `https://ubuntu.com/wsl/docs/latest/howto/install-ubuntu-wsl2/`

[8] Microsoft. *Set up a WSL development environment*. `https://learn.microsoft.com/en-us/windows/wsl/setup/environment`

[9] Microsoft. *Basic commands for WSL*. `https://learn.microsoft.com/en-us/windows/wsl/basic-commands`

[10] Microsoft. *Advanced settings configuration in WSL*. `https://learn.microsoft.com/en-us/windows/wsl/wsl-config`

[11] Microsoft. *Working across file systems*. `https://learn.microsoft.com/en-us/windows/wsl/filesystems`

[12] Free Software Foundation. *GNU Coreutils manual*. `https://www.gnu.org/software/coreutils/manual/coreutils.html`

[13] Free Software Foundation. *GNU Grep manual*. `https://www.gnu.org/software/grep/manual/grep.html`

[14] Python Software Foundation. *csv—CSV File Reading and Writing*. `https://docs.python.org/3/library/csv.html`

[15] Microsoft. *File Permissions for WSL*. `https://learn.microsoft.com/en-us/windows/wsl/file-permissions`

[16] Free Software Foundation. *Bash Reference Manual: Redirections*. `https://www.gnu.org/software/bash/manual/html_node/Redirections.html`

[17] Free Software Foundation. *Bash Reference Manual: Pipelines*. `https://www.gnu.org/software/bash/manual/html_node/Pipelines.html`

[18] Free Software Foundation. *Bash Reference Manual: Quoting*. `https://www.gnu.org/software/bash/manual/html_node/Quoting.html`

[19] Free Software Foundation. *Bash Reference Manual: The Set Builtin*. `https://www.gnu.org/software/bash/manual/html_node/The-Set-Builtin.html`

[20] Anaconda. *Miniconda: Linux installer*. `https://www.anaconda.com/docs/getting-started/miniconda/install/linux-install`; installer archive: `https://repo.anaconda.com/miniconda/`

[21] conda-forge contributors. *Miniforge*. `https://github.com/conda-forge/miniforge`

[22] Conda developers. *conda-libmamba-solver: User Guide*. `https://conda.github.io/conda-libmamba-solver/user-guide/`

[23] Mamba developers. *Mamba User Guide*. https://mamba.readthedocs.io/en/latest/user_guide/mamba.html

[24] Mamba developers. *Mamba Installation*. https://mamba.readthedocs.io/en/latest/installation/mamba-installation.html

[25] conda-forge contributors. *Transitioning from Anaconda's defaults channels*. https://conda-forge.org/docs/user/transitioning_from_defaults/

[26] Bioconda contributors. *Bioconda documentation: Usage*. https://bioconda.github.io/

[27] Conda developers. *Managing environments*. https://docs.conda.io/projects/conda/en/stable/user-guide/tasks/manage-environments.html

[28] Python Software Foundation. *The Python Tutorial*. https://docs.python.org/3/tutorial/

[29] Python Software Foundation. *The Python Tutorial: More Control Flow Tools*. https://docs.python.org/3/tutorial/controlflow.html

[30] Python Software Foundation. *The Python Tutorial: Input and Output*. https://docs.python.org/3/tutorial/inputoutput.html

[31] procps-ng contributors. *procps-ng: free, top and vmstat manuals*. https://gitlab.com/procps-ng/procps/-/tree/master/man

[32] Anshori M, Maulidiansyah, Sukron M. LLM inference performance analysis across Windows, Windows Subsystem for Linux 2 and native Linux. *JITK (Jurnal Ilmu Pengetahuan dan Teknologi Komputer)*. 2026;12(1):451–462. https://doi.org/10.33480/jitk.v12i1.8435

[33] Kinghorn D. *WSL2 vs Linux (HPL HPCG NAMD)*. Puget Systems; 2022. Technical report. https://www.pugetsystems.com/labs/hpc/wsl2-vs-linux-hpl-hpcg-namd-2354/

[34] Sobieraj M, Kotyński D. Docker performance evaluation across operating systems. *Applied Sciences*. 2024;14(15):6672. https://doi.org/10.3390/app14156672

[35] Conda developers. *conda run*. https://docs.conda.io/projects/conda/en/stable/commands/run.html