GiT ile Versiyon Kontrol Sistemi Eğitimi

Harezmi Bilişim Çözümleri

Versiyon Kontrol Sistemleri

- VCS nedir?
  - Uygulama kodunu yönetmemizi sağlayan sistemdir
  - Pratikte her tür dosya VCS'ler tarafından yönetilebilir
  - Dosyaların snapshot'ları versiyonlar olarak takip edilir
- Temel Özellikleri Nelerdir?
  - Geri dönüşlere imkan tanımaları (reversability)
  - Eş zamanlı çalışmaya izin vermeleri (concurrency)
  - Kod üzerinde notlar alınabilmesi (annotation)

Temel Özellikleri

- Geri dönüşe imkan tanıması
  - Bir hata yaptığımızda veya yaptığımız değişikliğin iyi bir fikir olmadığını anladığımızda kodun değişiklikten önceki haline dönebilmek önemlidir
- Eş zamanlı çalışmaya izin vermesi
  - Birden fazla kişinin uygulama kodu üzerinde aynı anda işlem yapmalarını sağlaması, bu işlemlerin düzenlenmesi ve birbirlerine etkilerinin yönetilmesi önemlidir

Temel Özellikleri

- Kod üzerine notlar alınabilmesi
  - Yapılan işlemlerin ne anlama geldiğini açıklayan ve anımsatan notlar ekip üyelerinin bir birleri ile iletişimine, kişinin yaptığı değişikliklerin nedenini hatırlamasına yardımcı olur

Temel Kabiliyetleri

- VCS'ler kabiliyetlerini sunmak için kodun "**master**" kopyasını tutarlar
- Geliştiriciler kendi **lokal kopya**ları (working copies) üzerinde işlem gerçekleştiriler
- Üç temel kabiliyete sahiptirler
  - Checkout/update
  - Checkin/commit
  - View history
- Diğer bütün işlemler özünde bu üç işlemin etrafında gerçekleşmektedir

Temel Kabiliyetleri

- Bütün VCSler bir **branch** kabiliyetine sahiptir
- Kodun **alternatif kollar**a ayrılarak farklı sürümler şeklinde paralel yönetilmesi sağlanır
- Bu kollar arasında birleştirme imkanı sunulur (**merge**)

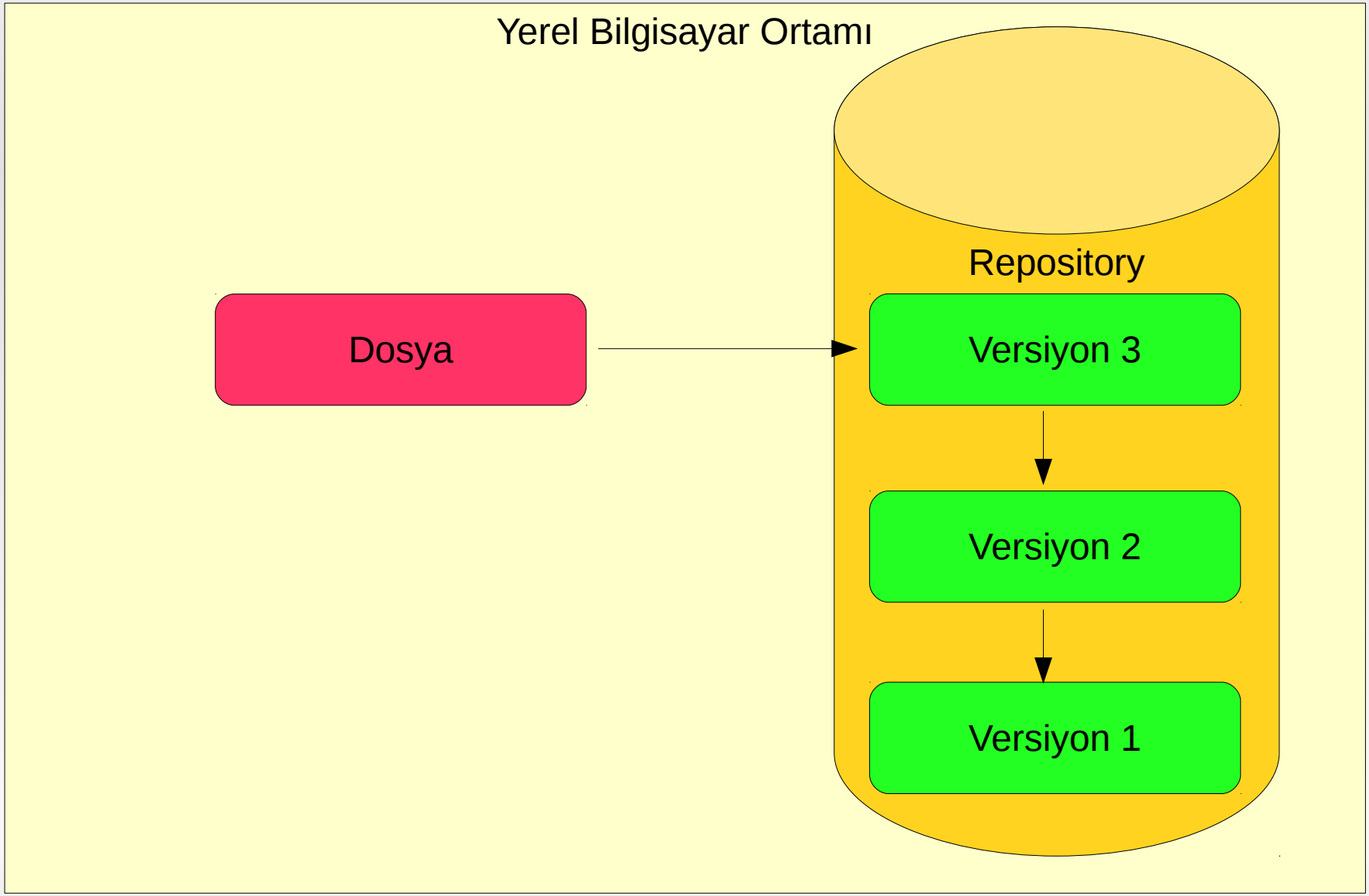
SCM ve VCS?

- Software Configuration Management daha kapsamlıdır
  - İçerisinde VCS barındırır
  - İlaveten projenin build işleminin yapılabilmesini sağlar
  - Bug ve issue takibi yapılmasını da sağlar

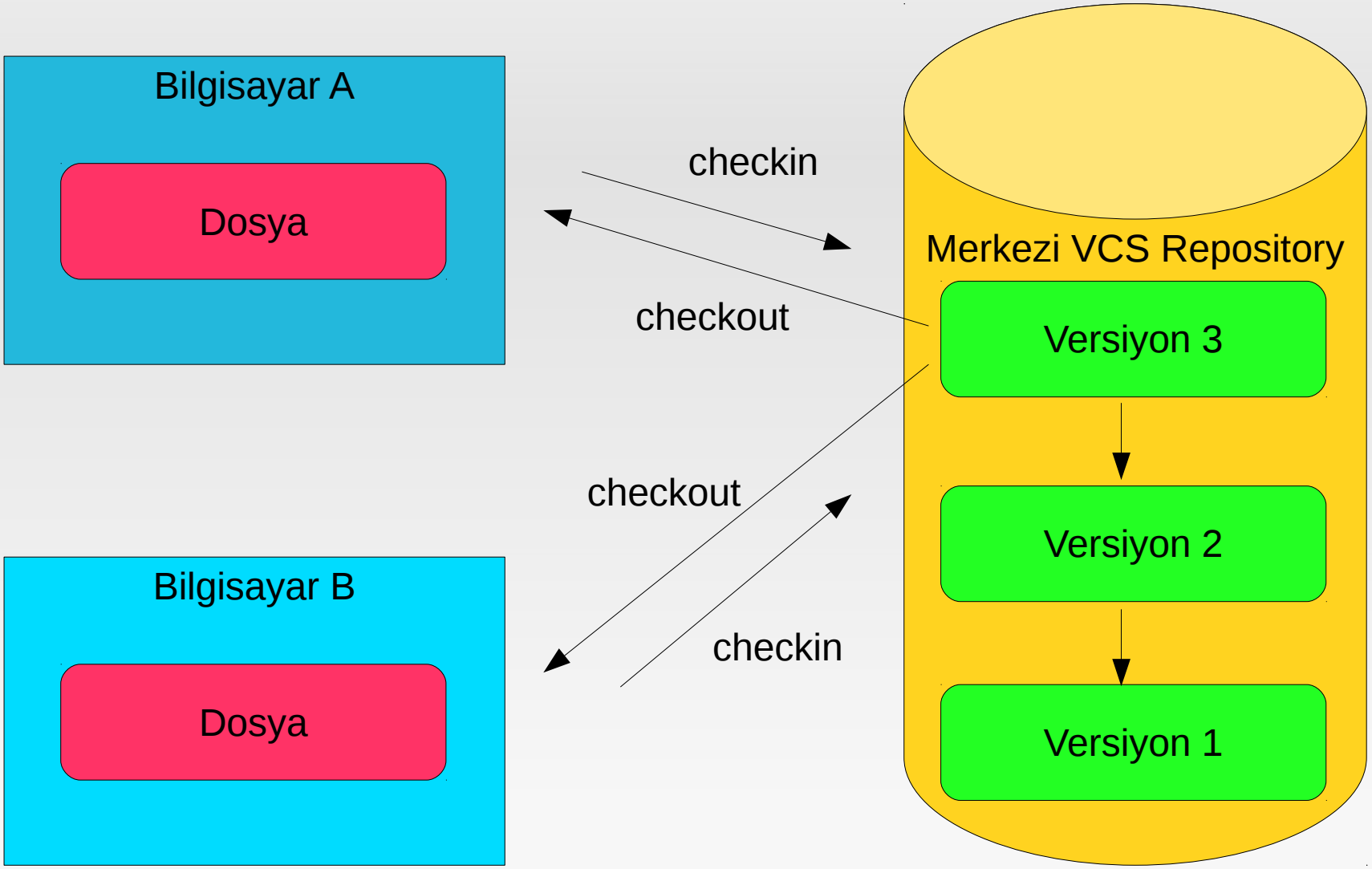
VCS'lerin Sınıflandırması

- Merkezi veya dağıtık mimaride olabilirler
- Commit öncesi veya sonrası merge yapabilirler
- Lock, changeset veya snapshot tabanlı olabilirler

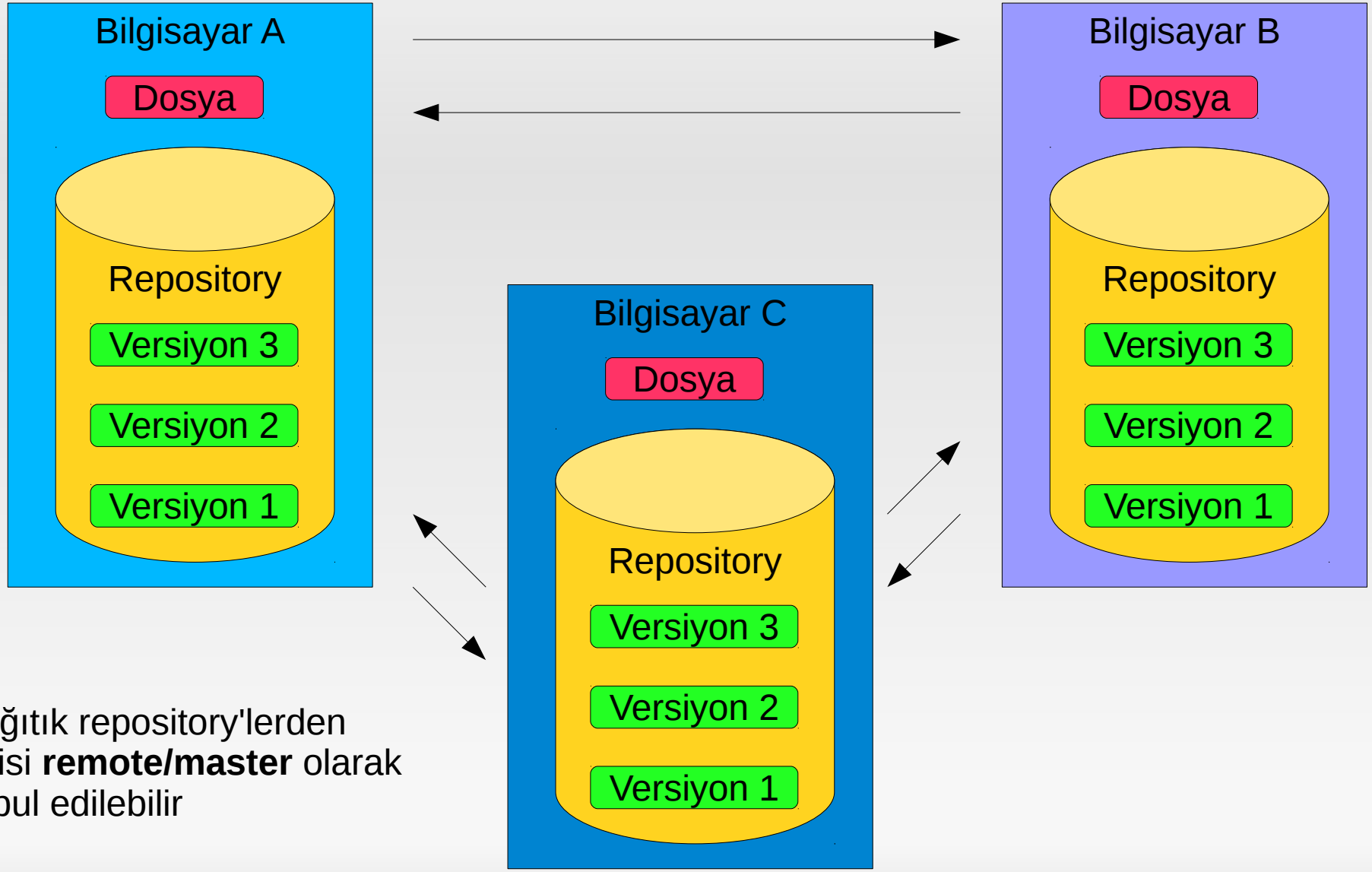
Local VCS Mimarisi



Merkezi VCS Mimarisi

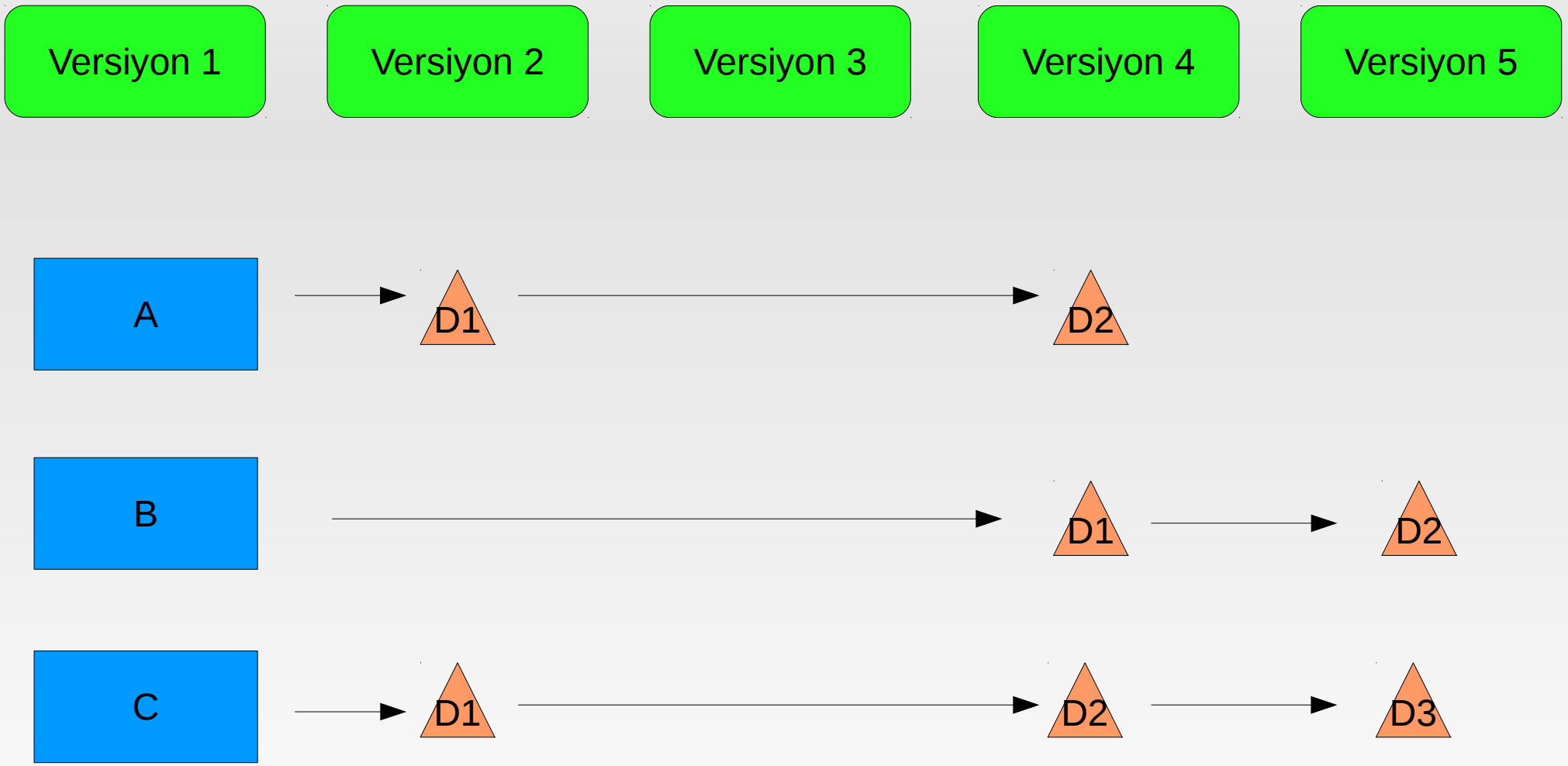


Dağıtık VCS Mimarisi

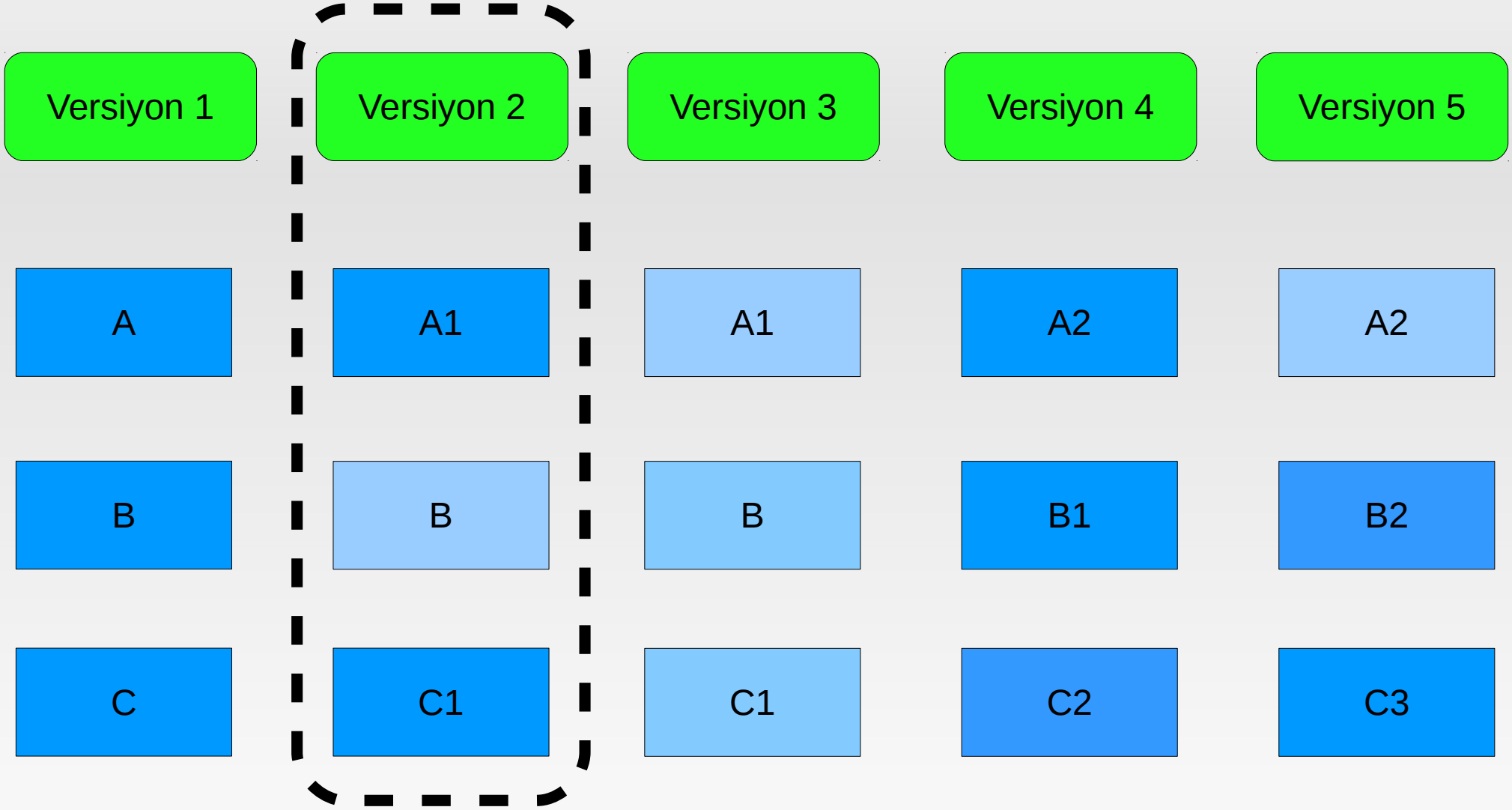


Dağıtık repository'lerden Birisi **remote/master** olarak kabul edilebilir

Changeset Yönetimi



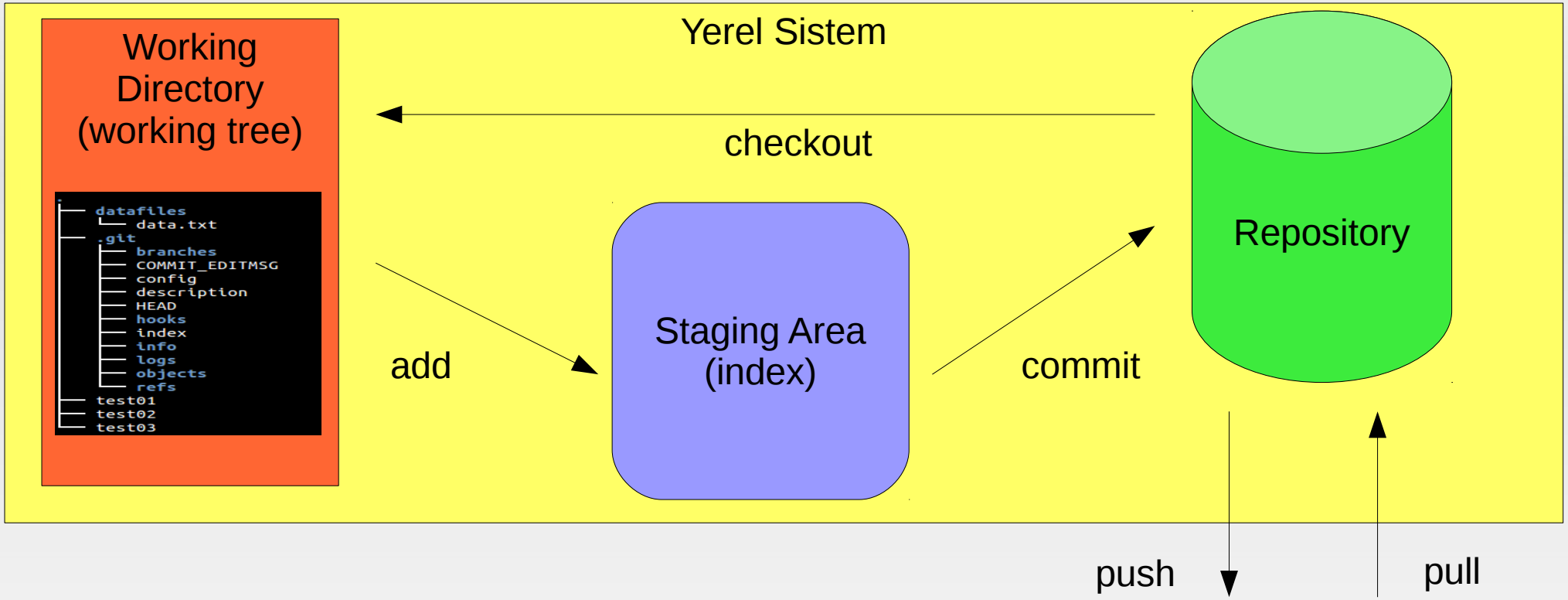
Snapshot Yönetimi



VCS'lerin Tarihsel Gelişimi

- Birinci Nesil
  - rcs
- İkinci Nesil
  - cvs
- Üçüncü Nesil
  - svn
- Dördüncü Nesil
  - git

GiT Mimarisi



Remote repository'den local repository'nin oluşturulma işlemine ise **clone** adı verilir

GiT ve Dosya State'leri

- **Untracked**: Stage alana eklenmemiş ve commitlenmemiş dosyaları belirtir
- **Tracked**: Commit'lenmiş ve stage alana atılmış dosyaları belirtir
- **Staged**: Bir sonraki commit işleminde yer alacağını belirtir
- **Dirty/modified**: Stage alana dahil edilmemiş dosyalardaki değişikliği anlatır

GiT Repository Tipleri

- **Non-bare (Normal) repository**
  - Working tree içeren repository'dir
- **Bare repository**
  - Working tree içermeyen repository'lere verilen isimdir
  - Genellikle server'daki remote repository working tree'ye ihtiyaç duymaz

Repository Oluşturma ve Clone İşlemleri

- `$git init repo`
- `$git init --bare repo`
- `$git clone https://github.com/harezmi/harezmi_test_repo remote-repo`
- `$git clone --bare https://github.com/harezmi/harezmi_test_repo remote-repo`

Repository Oluşturma ve Clone İşlemleri

- Remote repo clone işlemi, yerelde “**origin**” isimli bir **remote repo tanımı** eklenmesini sağlar. Ancak init komutu origin tanımını otomatik olarak eklemez
- `$git remote add origin ../remote-repo`
- `$git remote -v`
- `$git remote show origin`

Pull ve Push İşlemleri

- `$git push origin`
- `$git push origin master`
- `$git pull origin`
- Normalde sadece **bare repository'lere push** yapılabilir
- Ayrıca remote repolara yapılan push'ların **sadece fast forward merge** ile sonuçlanması gerekir
- Pull komutu `$git fetch` ve `$git merge/rebase` komutlarının kısa yoludur

GiT Konfigürasyonu

- Global veya repository düzeyinde konfigürasyon yapılabilir
- Global ayarlar **$USER_HOME/.gitconfig** dosyasında yönetilir
- Konfigürasyon ayarları `$git config` komutu ile yapılabilir
- `--global` opsiyonu ile yapılan ayar global konfigürasyona etki eder

GiT Konfigürasyonu

- $git config -- global user.name “Kenan Sevindik”
- $git cofig --global user.email “ksevindik@harezmi.com.tr”
- $git config --global core.editor vi
- $git config --global merge.tool kdiff3
- $git config --global list

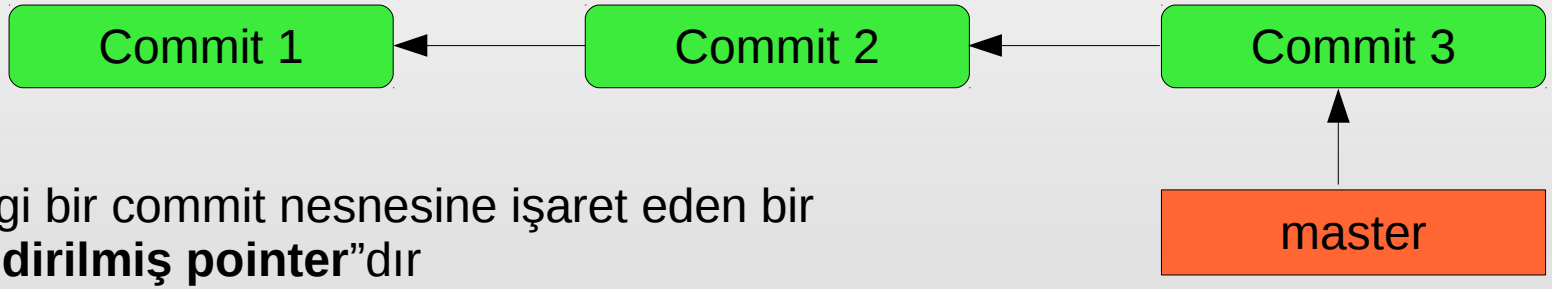
GiT Commit Nesneleri

- Yapılan her commit için ayrı bir “**commit object**” oluşturulur
- Commit object repository'deki sürüme karşılık gelen snapshot verisine doğrudan erişim imkanı sunar
- Commit object **SHA-1 checksum** değeri ile adreslenir
- SHA-1 checksum değeri yeni commit'e kadar verideki değişikliklerden, committer ve diğer bazı verilerden oluşturulur

GiT Commit Nesneleri

- Her commit nesnesi kendinden bir önceki commit nesnesini bilir
- Committer ve author bilgileri ayrı ayrı takip edilir
- **Tree object ID** değerleri tutulur
- Tree Object ID değeri bu commit içinde yer alan her bir dosyaya ayrı ayrı erişmeyi sağlar

GiT ve Branch Kavramı

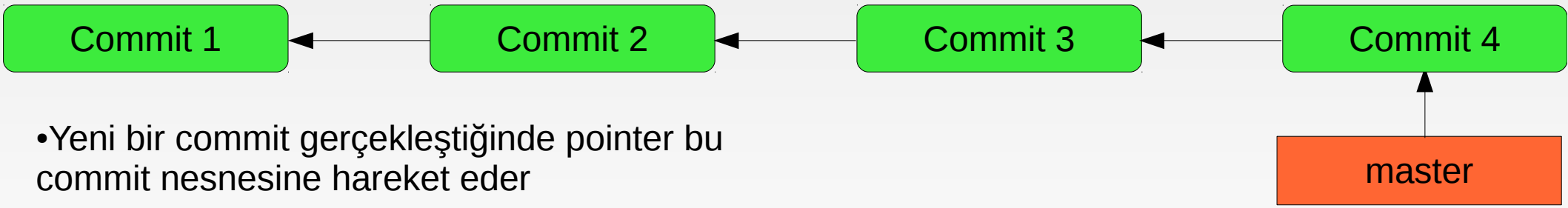


•Herhangi bir commit nesnesine işaret eden bir "**isimlendirilmiş pointer**"dır

•Bir repository "**clone**" yapılırken default bir branch oluşturulur

•Default branch'a "**master**" adı verilir

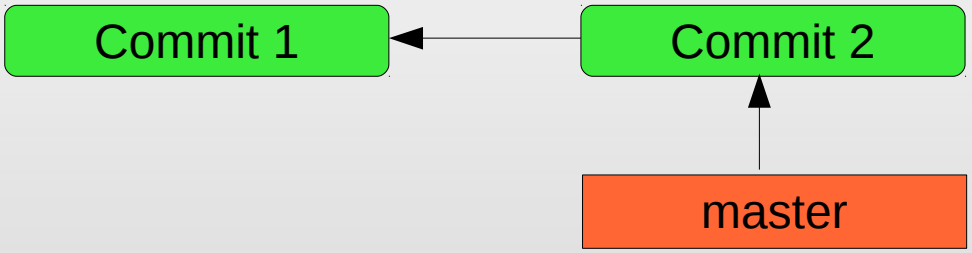
```
$git commit -m "i did some changes"
```



•Yeni bir commit gerçekleştiğinde pointer bu commit nesnesine hareket eder

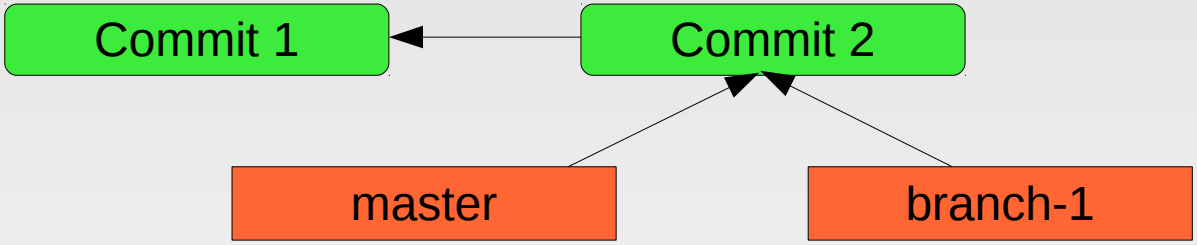
•Bir branch'tan başka bir branch daha oluşturulabilir

Yeni Branch Oluşturma

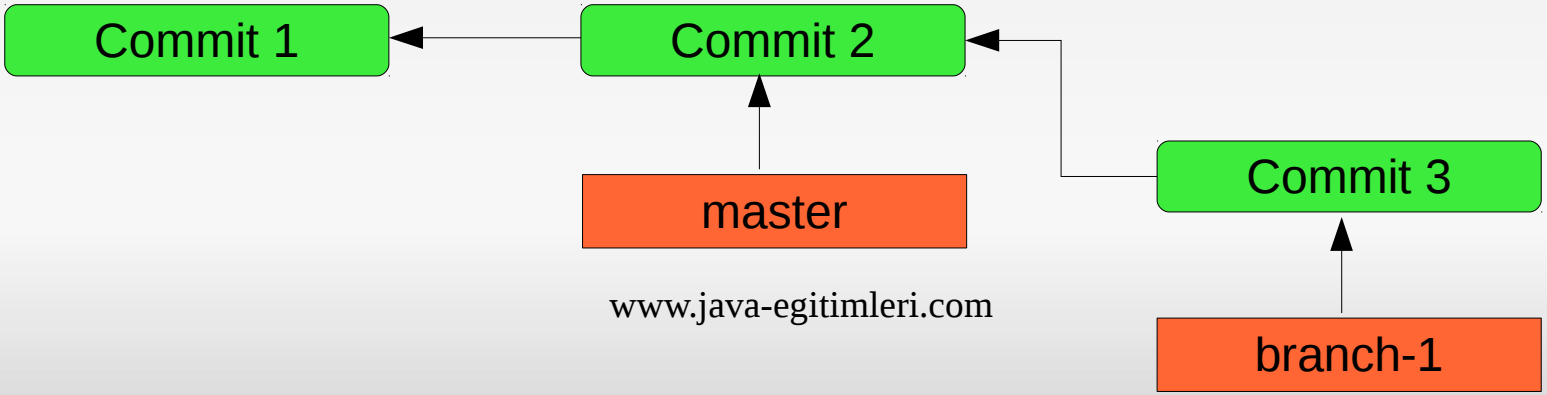



```
$git branch branch-1
```

```
$git checkout branch-1
```



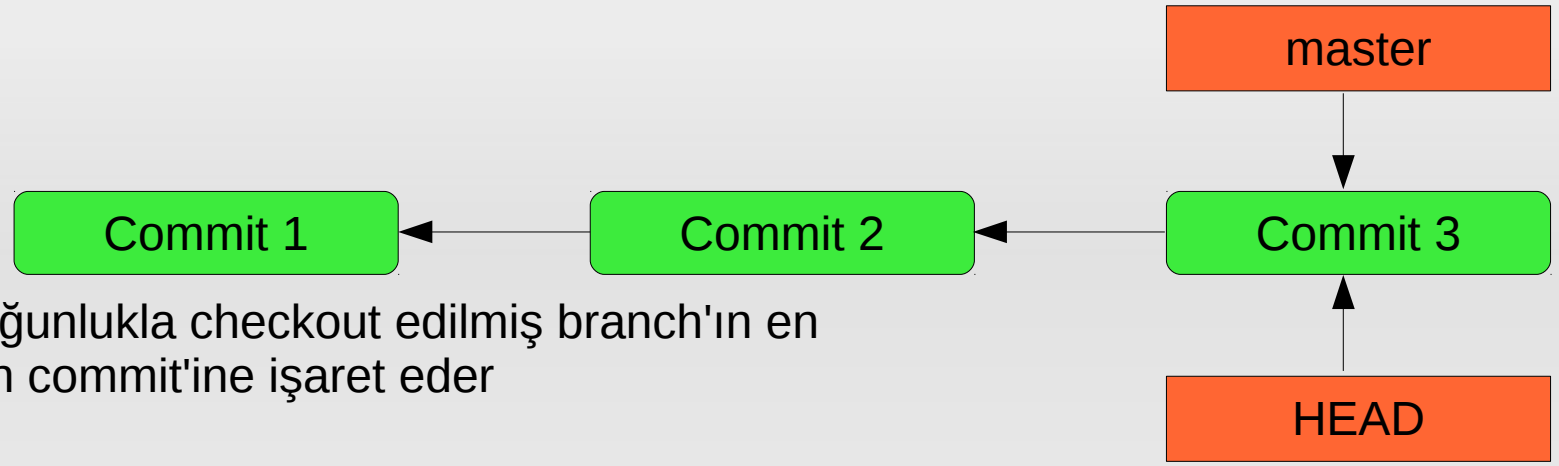

```
$git commit -m “my changes on branch-1”
```



Branch İşlemleri

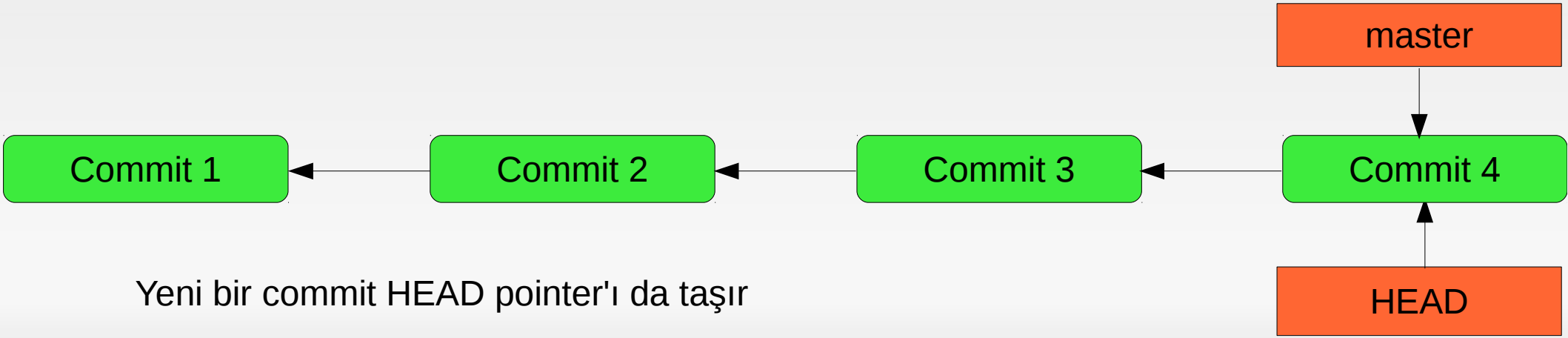
- `$git branch`
- `$git branch -a`
- `$git branch -b branch-2` :komutu yeni branch'ı oluşturur ve onu checkout eder
- `$git branch -m branch-old-name branch-new-name`
- `$git branch -d branch-2` :komutu belirtilen branch'ı siler
- `$git diff master branch-1`

GiT HEAD



Çoğunlukla checkout edilmiş branch'ın en son commit'ine işaret eder

```
$git commit -m “i did some changes”
```



Yeni bir commit HEAD pointer'ı da taşır

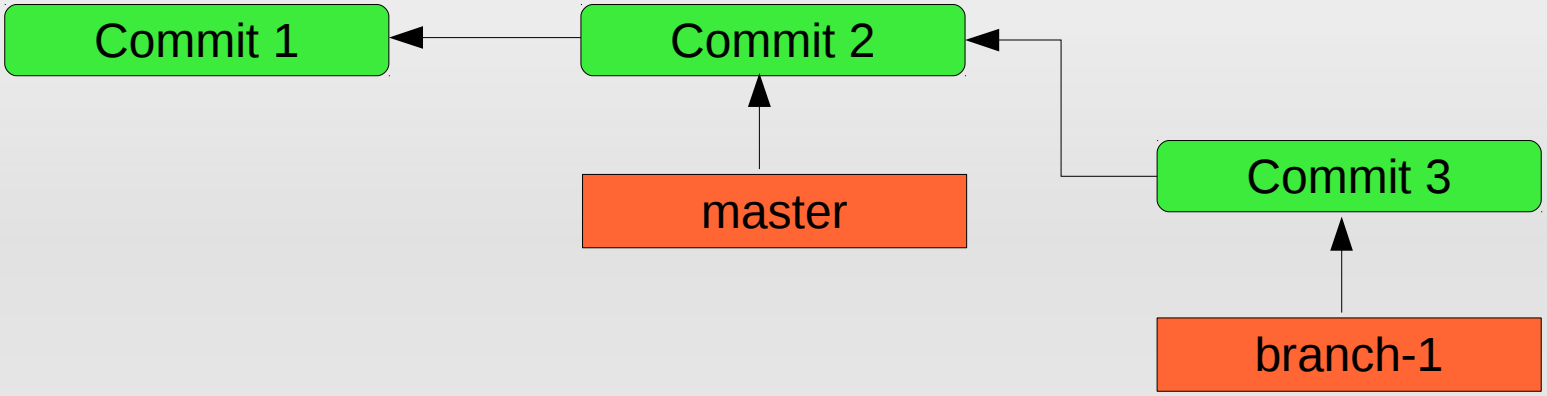
GiT HEAD

- Bazı durumlarda herhangi bir başka commit nesnesine işaret ediyor olabilir (detached head mode)
- Bu durumda yapılan commit HEAD pointer'ı hareket ettirmez
- Spesifik bir commit nesnesi checkout edildiği vakit ise HEAD pointer bu commit nesnesini gösterir

Merge İşlemleri

- Fast forward merge
- Merge commit

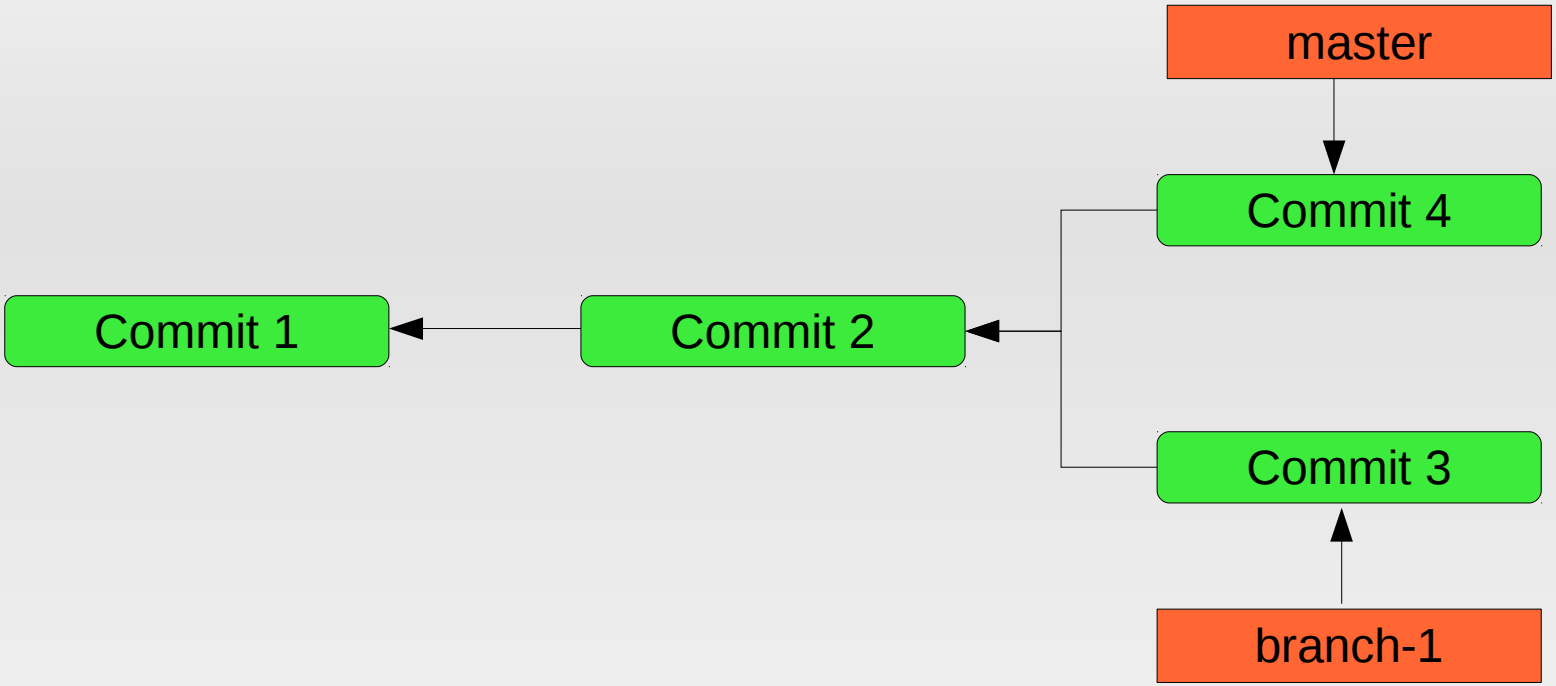
Fast Forward Merge




```
$git checkout master
$git merge branch-1
```

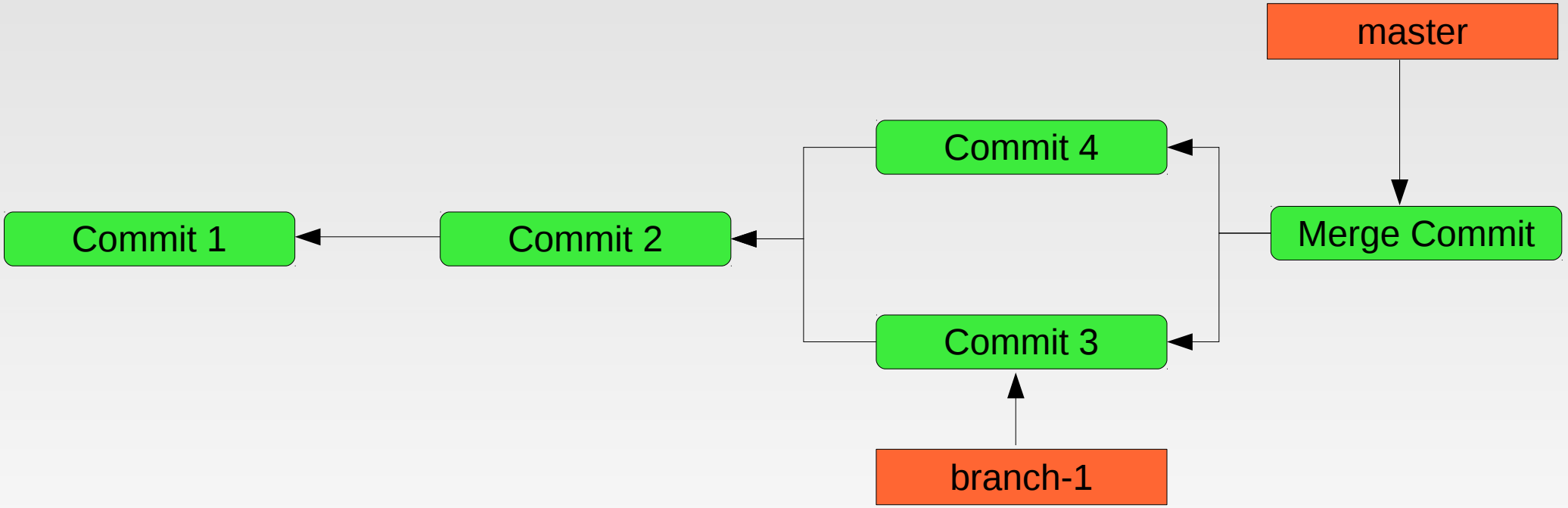
Fast forward merge sadece pointer'ı taşır

Merge Commit - Öncesi

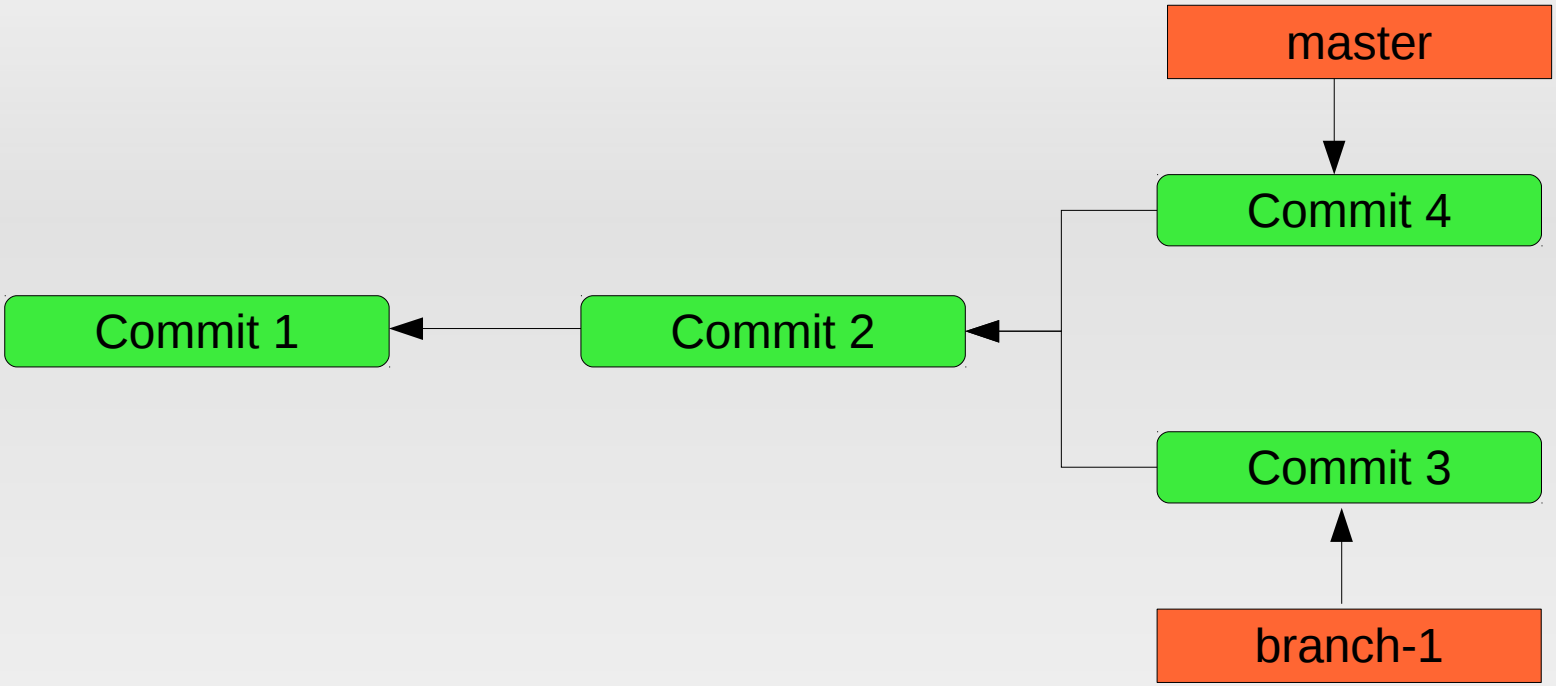



```
$git checkout master
$git merge branch-1
```

Merge Commit - Sonrası

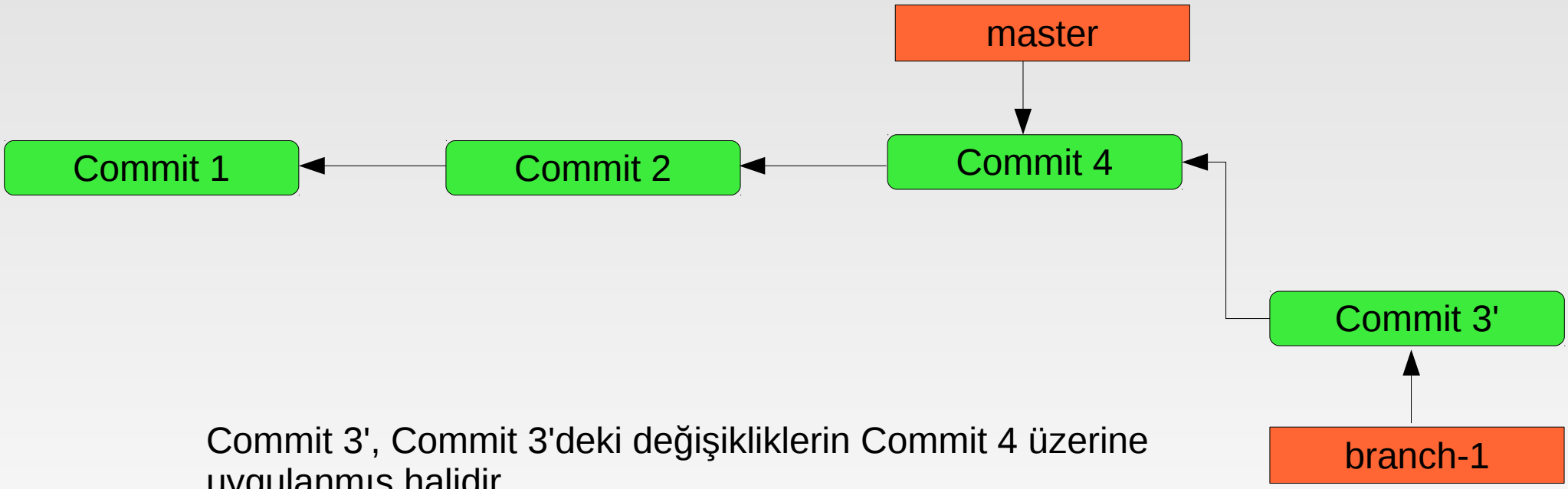


Rebase İşlemi - Öncesi




```
$git checkout branch-1
$git rebase master
```

Rebase İşlemi - Sonrası



Commit 3', Commit 3'deki değişikliklerin Commit 4 üzerine uygulanmış halidir

Tag İşlemleri

- Herhangi bir commit nesnesinin isimlendirilmesidir
- Bu sayede commit nesnesine erişim daha kolay olur
- İki türlü tag'leme yapılabilir
  - Lightweight
  - Annotated

Tag İşlemleri

- Lightweight tag sadece isimden oluşur
- Annotated tag ise ilave bilgiler taşıyabilir
  - Tag'i oluşturan kişinin isim ve e-posta bilgileri
  - Tag mesajı
  - Tag oluşturma zamanı
- Annotated tag'lerin imzalanması ve verify edilmesi de mümkündür
  - GNU Privacy Guard

Tag İşlemleri

- `$git tag 1.0.0`
- `$git tag 1.0.0 -m “Release version 1.0.0”`
- `$git tag 1.0.0 <commit-id>`
- `$git show 1.0.0`
- `$git checkout 1.0.0`
- `$git push origin 1.0.0` :önce 1.0.0 isimli bir branch varsa onu push edecektir
- `$git push origin tag 1.0.0`
- `$git tag -d 1.0.0`

Repository Hakkında Bilgi Edinme

- `$git status` :repository'nin halihazırdaki durumu hakkında bilgi verir ve önerilerde bulunur
- `$git log` :Halihazırdaki branch'da yapılan commit'leri listeler
- `$git log a.txt` :Dosya üzerindeki commit'leri listeler
- `$git log -p a.txt` :Her bir commit arasındaki farkı listeler
- `$git log -- a.txt` : Değişiklikleri dosya silinmiş olsa bile listeler

Repository Hakkında Bilgi Edinme

- `$git diff` : Stage ile working tree arasındaki farkları listeler
- `$git diff --cached` : Son commit ile stage arasındaki farkları listeler
- `$git show <commit-id>` : Belirtilen commit nesnesinin değişikliklerini listeler
- `$git diff HEAD~1 HEAD` : İki commit nesnesi arasındaki farkları listeler
- `$git blame a.txt` :Belirtilen dosya ve dizin üzerinde kimin işlem yaptığını listeler

GiT Stash İşlemi

- Stash işlemi working tree ve staging area'daki değişikliklerin saklanıp en son revision'a geri dönülmesini sağlar
- Bu sayede acil bir bug fix gerçekleştirilip tekrar çalışmalara kalınan yerden devam edilebilir
- Birden fazla stash'in yönetilmesi de mümkündür
- Mevcut stash nesneleri listelenebilir, geri dönüşte kullanılabilir veya silinebilir

GiT Stash İşlemi

- `$git stash` : Commit edilmemiş değişikliklerden bir stash oluşturur
- `$git stash pop` : Stash edilmiş değişiklikleri aktive eder ve stash'i listeden çıkarır
- `$git stash save` :Commit edilmemiş değişiklikleri stash listesine ekler
- `$git stash list` :Stash listesini gösterir

GiT Stash İşlemi

- `$git stash apply stash@{0}`:Belirtilen stash nesnesini aktive eder
- `$git stash drop stash@{0}` :Belirtilen stash nesnesini siler
- `$git stash clear` :Stash listesini temizler
- `$git stash branch branch-name` : Stash'deki değişiklikleri kullanarak yeni bir branch oluşturur

Değişiklikleri Geri Alma

- `$git clean` : Track edilmeyen dosyaları siler
- `$git clean -n`: Hangi track edilmeyen dosyaların silineceğini gösterir
- `$git clean -f -d -x` :
  - -f: clean işlemini zorlar
  - -d:dizinleri siler
  - -x:gizli dosyaları siler

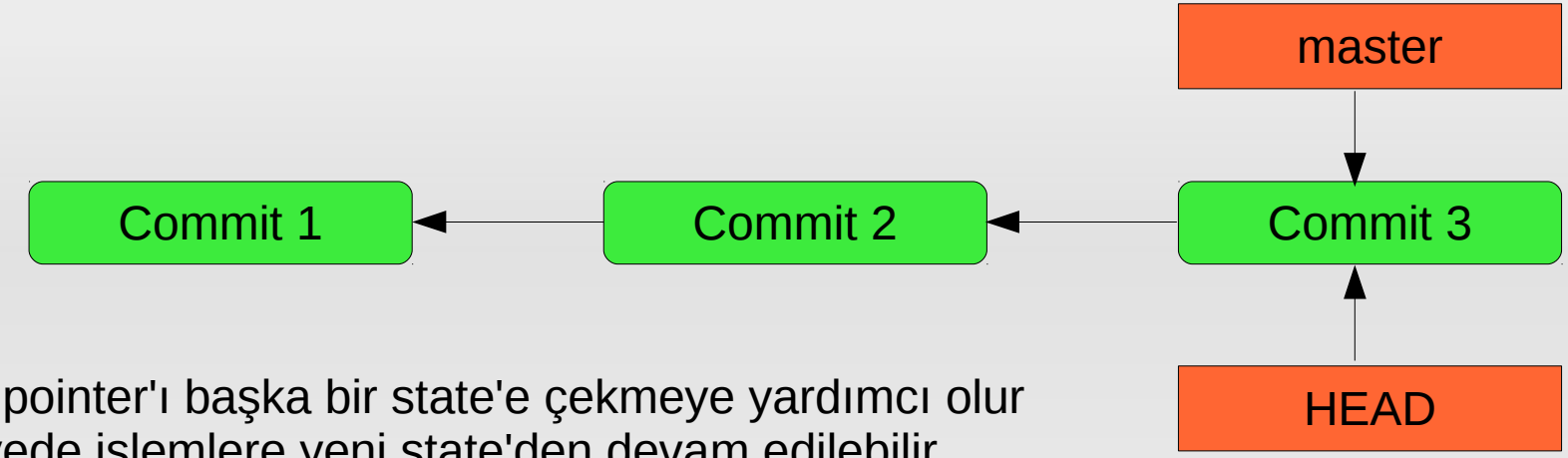
Değişiklikleri Geri Alma

- `$git checkout <unstaged-path>` : Stage edilmemiş değişikliği geri alır
- `$git checkout -- <unstaged-path>` : Stage edilmemiş silme işlemini veya değişikliği geri alır
- `$git checkout -- <modified-dir>` : Commit edilmemiş dizin silme işlemini geri alır
- `$git checkout HEAD -- <staged-path>` : Stage edilmiş değişiklik/silme işlemini geri alır

Değişiklikleri Geri Alma

- `$git reset <staged-file>`: Stage edilmiş değişikliği stage'den çıkarır
- `$git reset --hard` : working tree'yi HEAD ile aynı yapar, fakat track edilmeyen dosyaları silmez
- `$git revert <commit-id>` :Belirli bir commit nesnesine revert eder

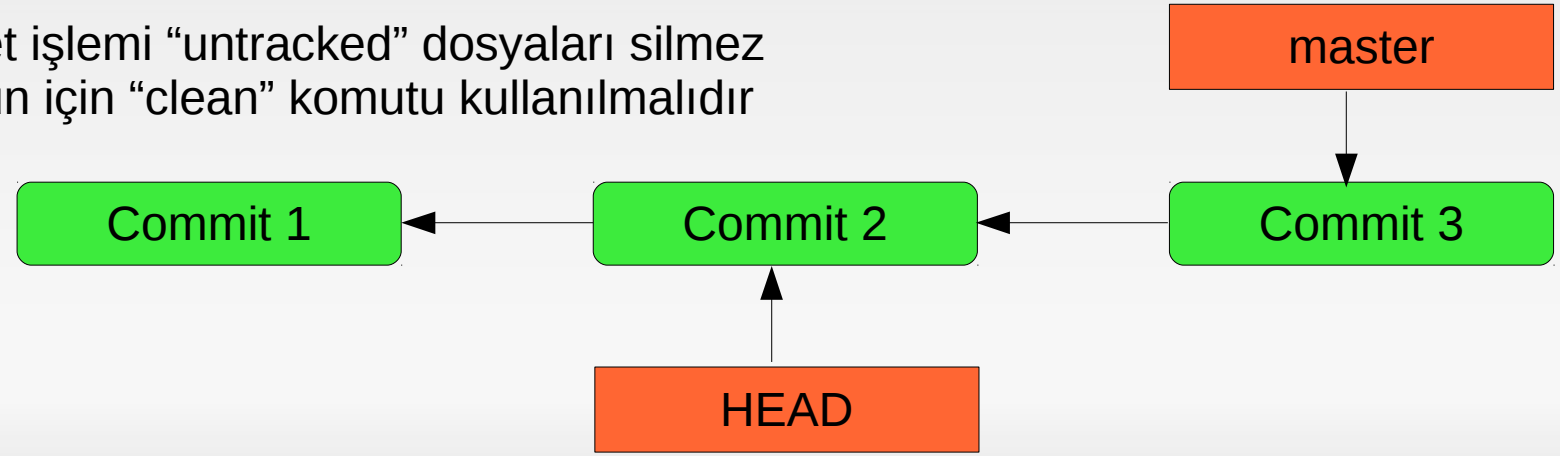
HEAD Reset İşlemi



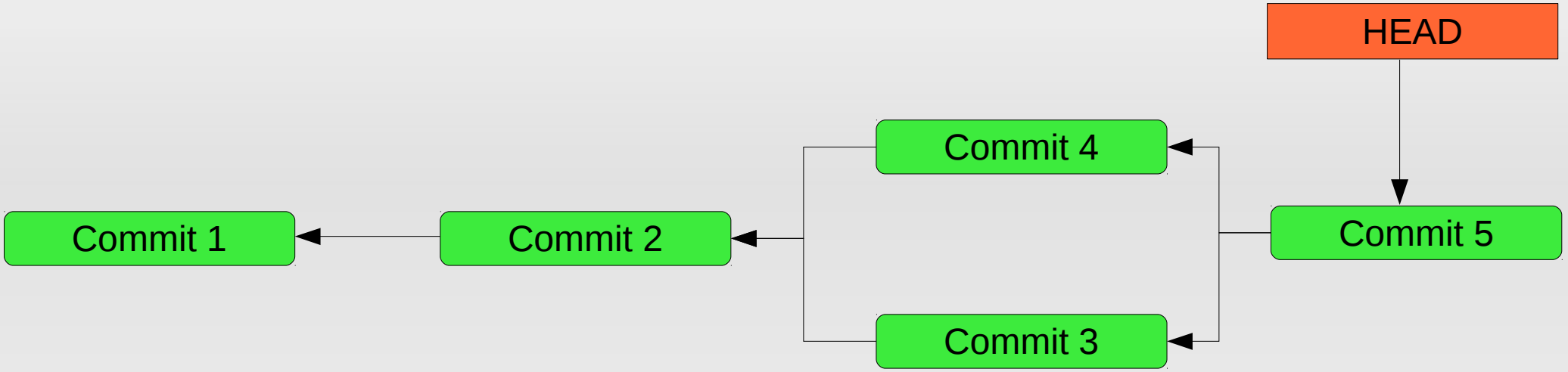
HEAD pointer'ı başka bir state'e çekmeye yardımcı olur
Bu sayede işlemlere yeni state'den devam edilebilir

```
$git reset HEAD~1
```

Reset işlemi “untracked” dosyaları silmez
Bunun için “clean” komutu kullanılmalıdır



~ ve ^ Kullanımı



- HEAD~1: Commit 4'ü gösterir
- HEAD~2: Commit 2'yi gösterir
- HEAD^1: Commit 4'ü gösterir
- HEAD^2: Commit 3'ü gösterir

Reset Seçenekleri

RESET	HEAD	Working tree	Staging area
soft	EVET	HAYIR	HAYIR
mixed*	EVET	HAYIR	EVET
hard	EVET	EVET	EVET

*: default değerdir

Double ve Triple Dot Operatörleri

- `$git log HEAD~4..HEAD` : head ve head~4 arasındaki bütün commit'leri gösterir
- `$git log testing..master` : master'da olan fakat testing branch'da olmayan commit'leri gösterir
- `$git log master..testing` :testing branch'da olup, master'da olmayan commit'leri gösterir
- `$git log testing...master` :her ikisinde de olmayan, fakat sadece birinde olan commit'leri gösterir

Patch İşlemleri

- $git format-patch origin/master : En son commit mesajını kullanarak bir patch dosyası oluşturur
- $git apply <patch-file>
- $git format-patch -1 HEAD
- $git format-patch -3 HEAD

İletişim

- Harezmi Bilişim Çözümleri
- Kurumsal Java Eğitimleri
- http://www.java-egitimleri.com
- info@java-egitimleri.com