Git Sürüm Kontrol Sistemi ve Uygulamalı Örnekler

ZAFER GÜREL

16 NİSAN 2019 – SAKARYA CODERS SUNUMU

Git Nedir?

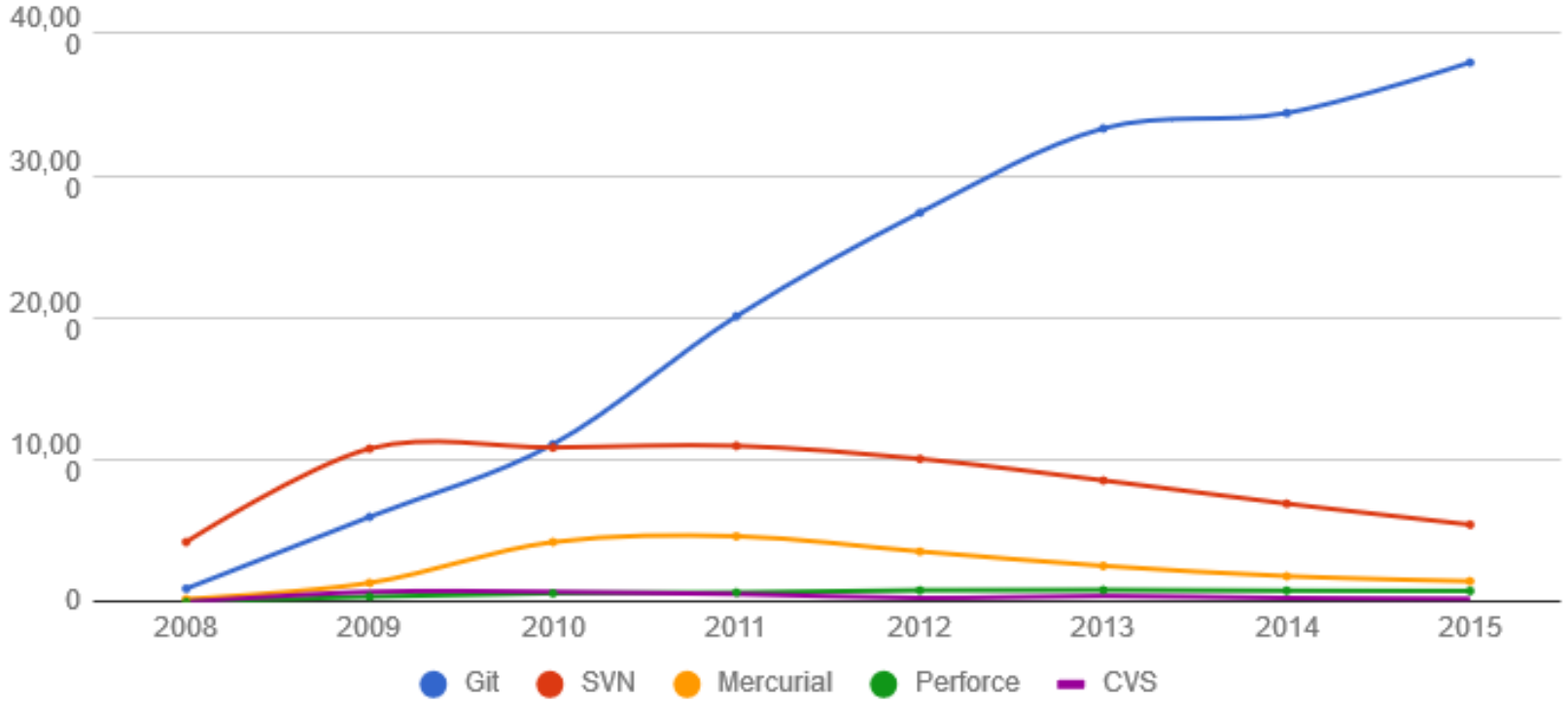
Git, hız ve verimlilik için tasarlanmış açık kaynak kodlu, dağıtık sürüm kontrol sistemidir.

2005 yılında, BitKeeper firması, Linux çekirdeğini geliştiren ekibine sunduğu ücretsiz sürüm kontrol sistemi lisansını iptal etti.

Bunun üzerine 2005 yılında, çekirdeği geliştiren ekip ve özellikle Linus Torvalds, BitKeeper tecrübesini de dikkate alarak kendi araçlarını geliştirmeye başladı.

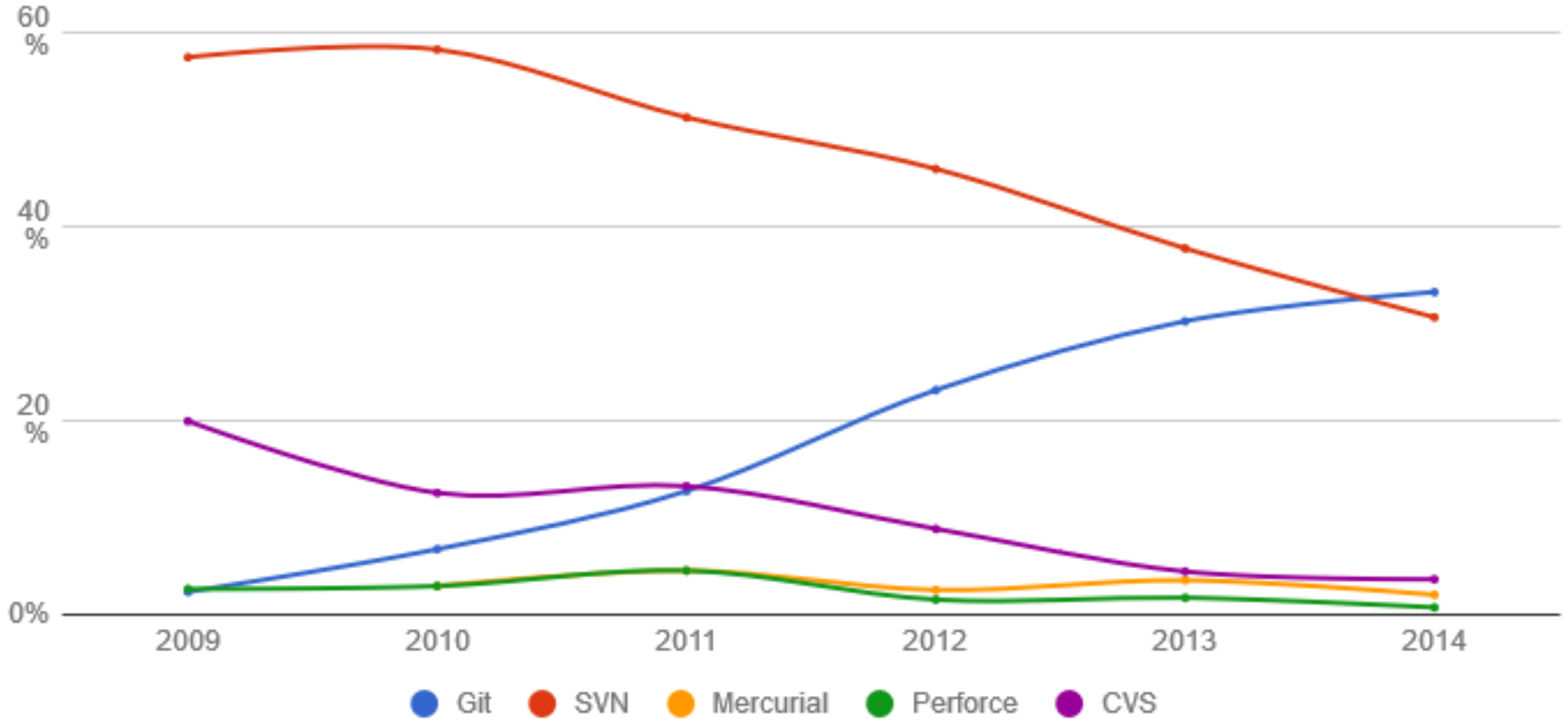
Böylece hayata merhaba diyen Git sistemi, yıllar içerisinde olgunlaşarak çok hızlı, büyük projelerde çok verimli ve özellikle çok basit ve hızlı şekilde dal oluşturma (branching) özelliğiyle doğrusal olmayan geliştirmeye olanak tanıyacak hale geldi.

Stack Overflow'daki Soru Sayıları (2016)



https://rhodecode.com/insights/version-control-systems-2016

Eclipse Topluluğu Geliştiricilerinin Kullandığı Sistemler



https://rhodecode.com/insights/version-control-systems-2016

Neden Git?

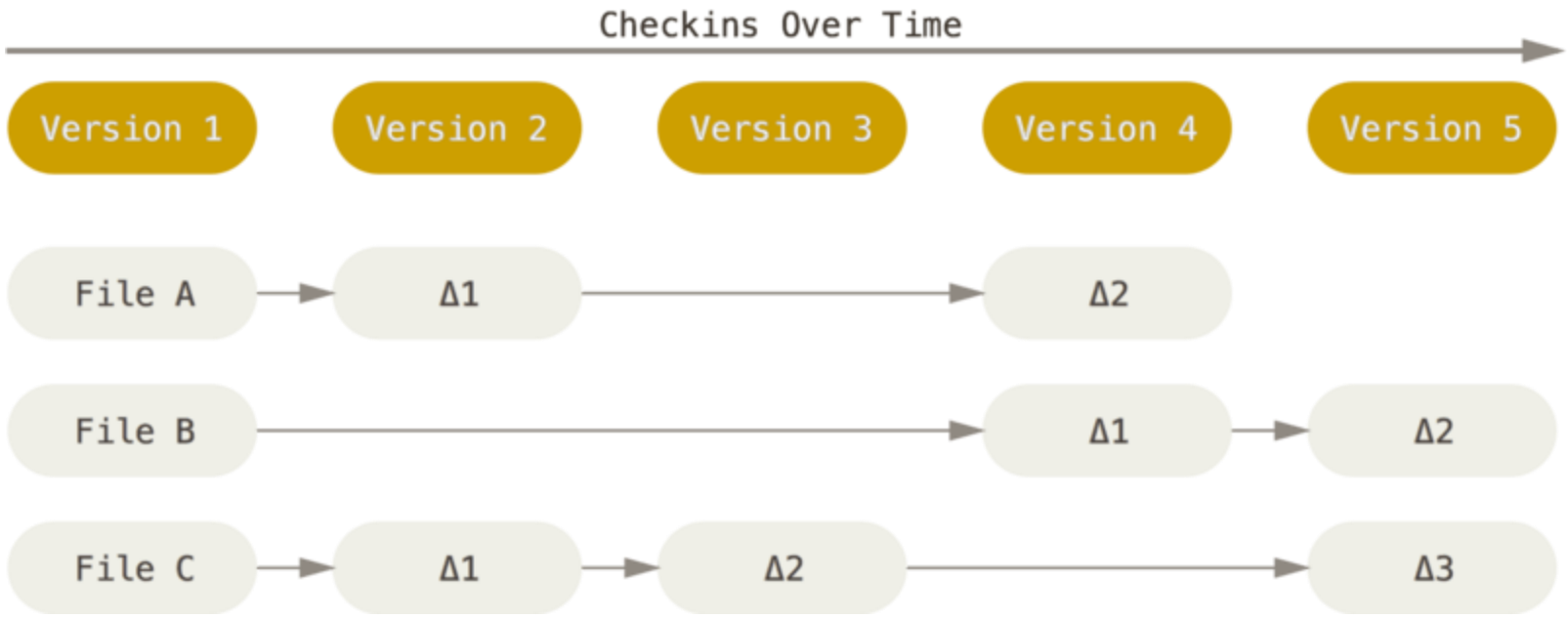
Her şey lokalde – Tüm proje geçmişi dahil

Çok hızlı

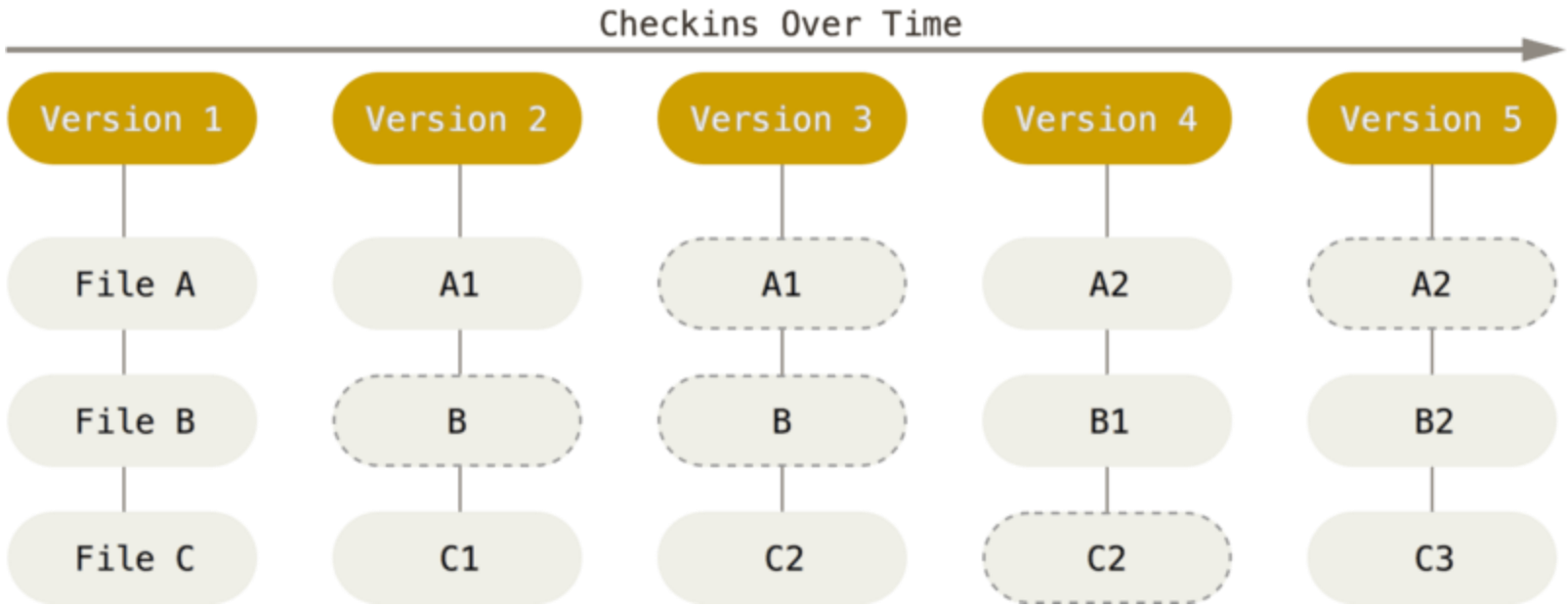
Her değişikliğin bir kopyası (snapshot) saklanır, farklar değil

Merkezi değil, dağıtık. Her şey (proje tarihçesi, sürümler) herkeste.

Diğer Sürüm Kontrol Sistemleri



Git’in Sürümleri Saklama Şekli



Git Kurulumu

Linux, macOS ve Windows'a kurulabilir.

Linux
sudo dnf install git-all (Fedora, RHAL veya CentOS)
sudo apt install git (Debian / Ubuntu)

Windows
https://git-scm.com/download/win

MacOs
https://git-scm.com/download/mac.

Kaynak kod ile kurulum
https://git-scm.com/book/en/v2/Getting-Started-Installing-Git

Öğrenme Kaynakları

Her platform için Git, ücretsiz çevrimiçi Git Kitabı

www.git-scm.com

GitHub'da Git öğrenme kaynakları

https://try.github.io/

Atlassian'dan Öğrenme Kaynakları

https://confluence.atlassian.com/bitbucketserver061/git-resources-968674263.html

Favori Git Komutlarım

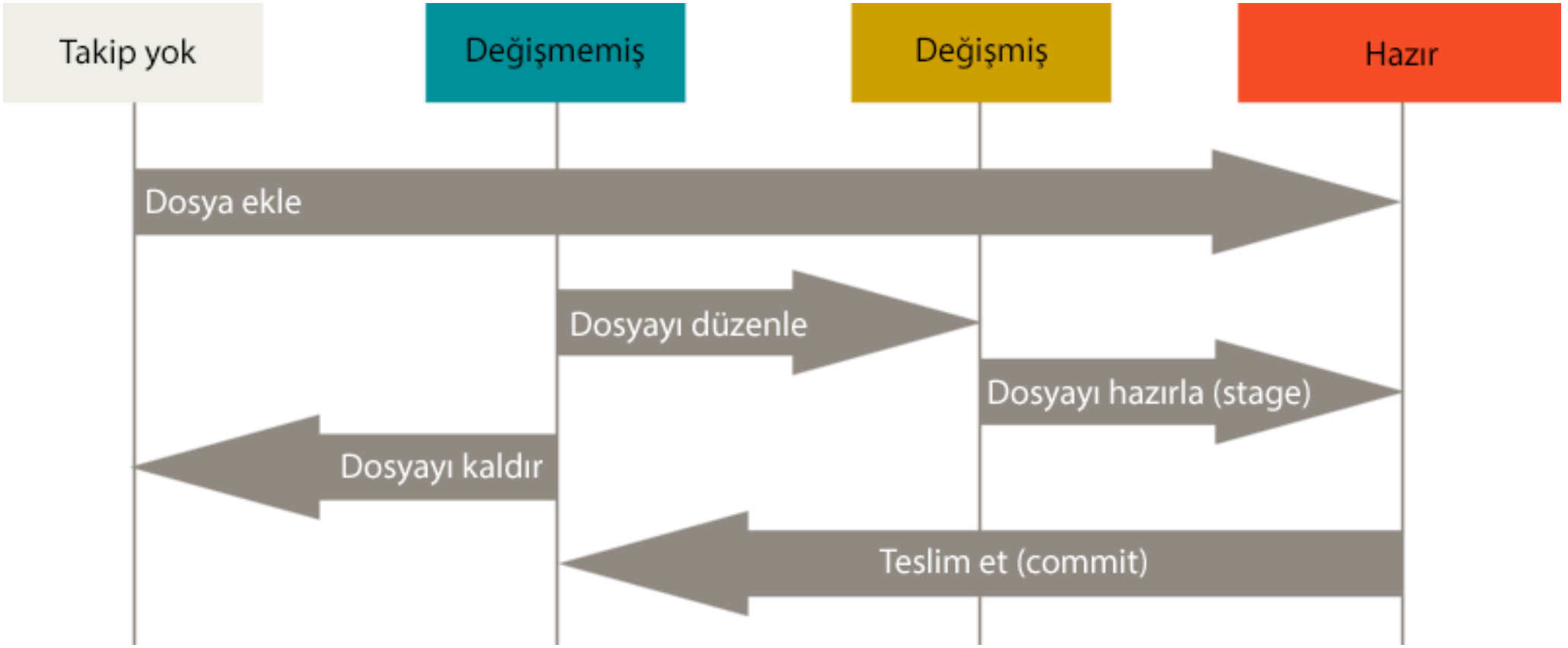
https://github.com/zafergurel/favori-git-komutlari/

İlk Ayarlar

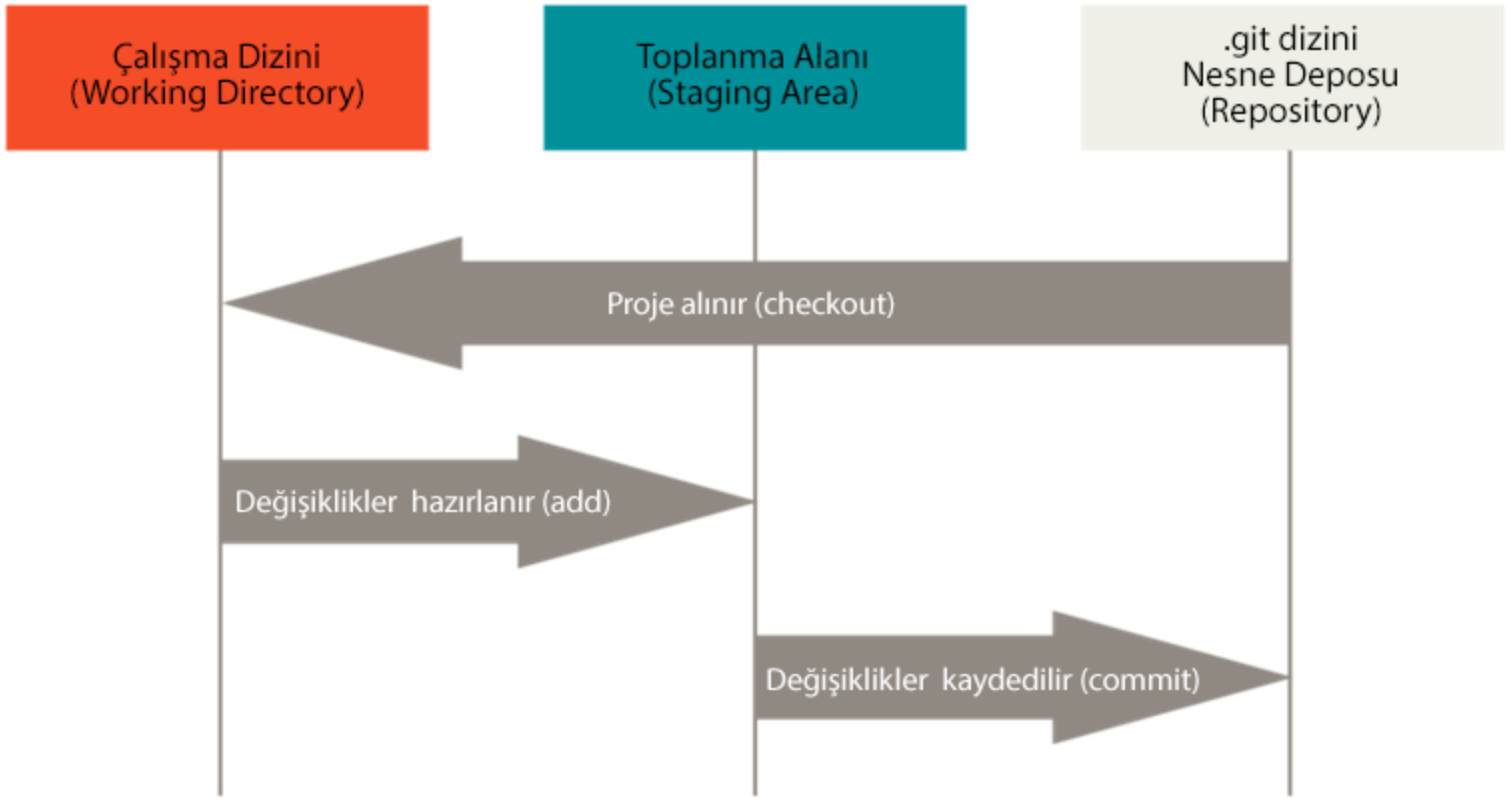
Kişisel Bilgileri Girme	Ayarları Görme
Adınız git config --global user.name "Ahmet Çınar " **E-posta** git config --global user.name ahmet@cinarltd.com	**Ayarları kontrol etmek için:** git config --list (Tüm ayarlar) git config user.name (Tek ayar)

Ayarları editör ile düzenleme	Hangi ayar hangi dosyada?
git config --global --edit git config --system --edit git config --local --edit	git config --list --show-origin

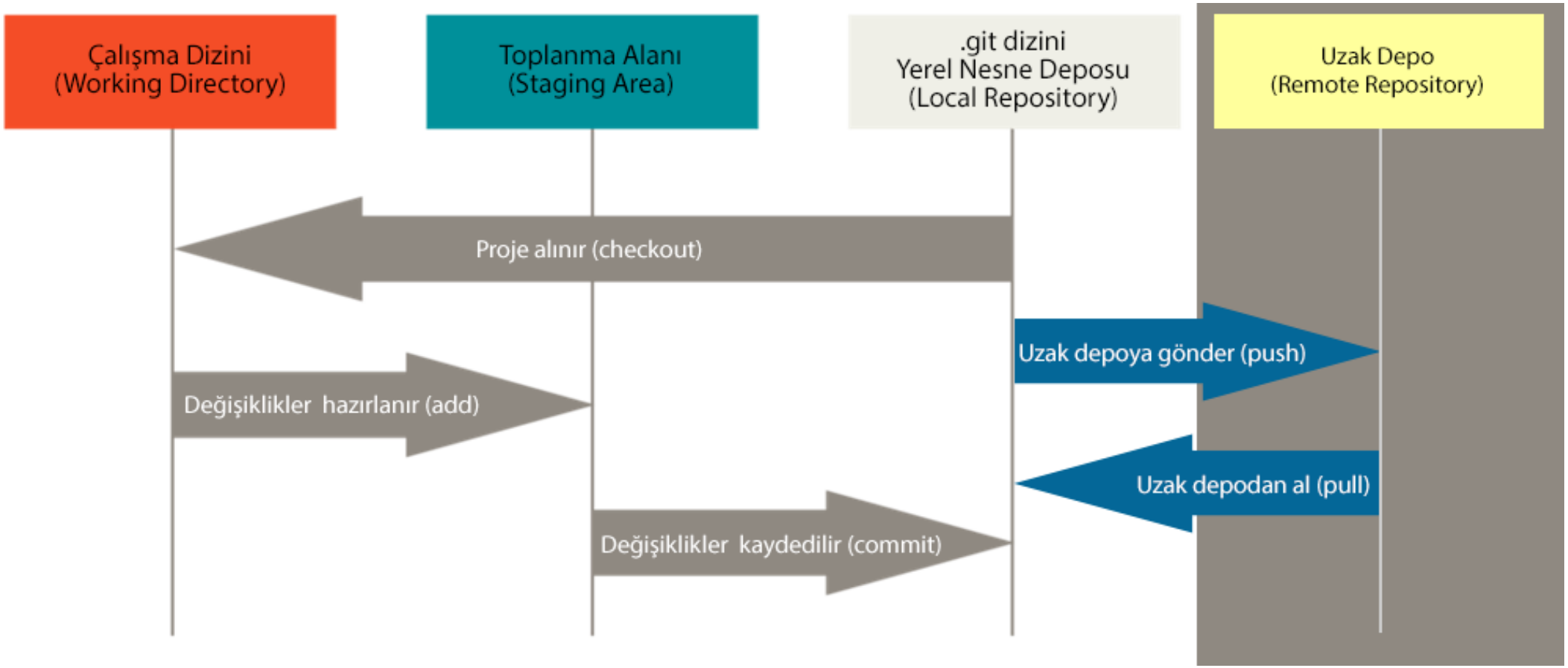
Temel Akış


Takip yok
Değişmemiş
Değişmiş
Hazır
Dosya ekle
Dosyayı düzenle
Dosyayı hazırla (stage)
Dosyayı kaldır
Teslim et (commit)

Üç Ana Bölüm


Çalışma Dizini
(Working Directory)
Toplanma Alanı
(Staging Area)
.git dizini
Nesne Deposu
(Repository)
Proje alınır (checkout)
Değişiklikler hazırlanır (add)
Değişiklikler kaydedilir (commit)

Üç Ana Bölüm ve Uzak Depo



Git Nesneleri

Her nesne, SHA1 algoritması ile hesaplanan 40 karakterden (128 bit) oluşan «nesne isimleri» ile isimlendirilir.

6ff87c4664981e4397625791c8ea3bbb5f2279a3

Bu sayede, Git nesnelerin aynı olup olmadıklarını sadece isimleri karşılaştırarak hızlıca anlayabilir. İsterse dosyanın büyüklüğü 100 MB olsun.

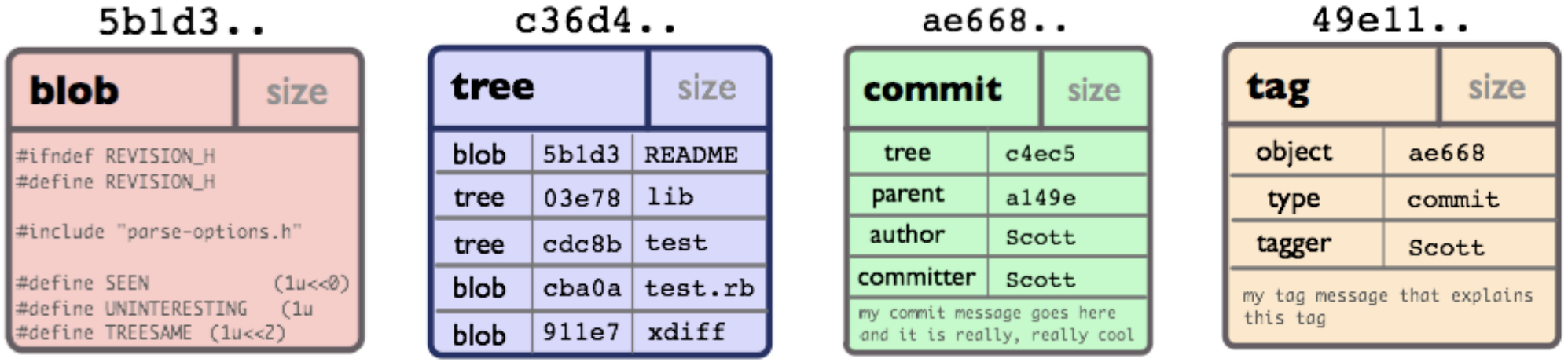
SHA1 fonksiyonu, kriptografik özüt fonksiyonudur ve iki farklı nesnenin aynı ismi alması imkansız olmasa* bile pratikte çok zordur.

*: https://www.theregister.co.uk/2017/02/23/google_first_sha1_collision/

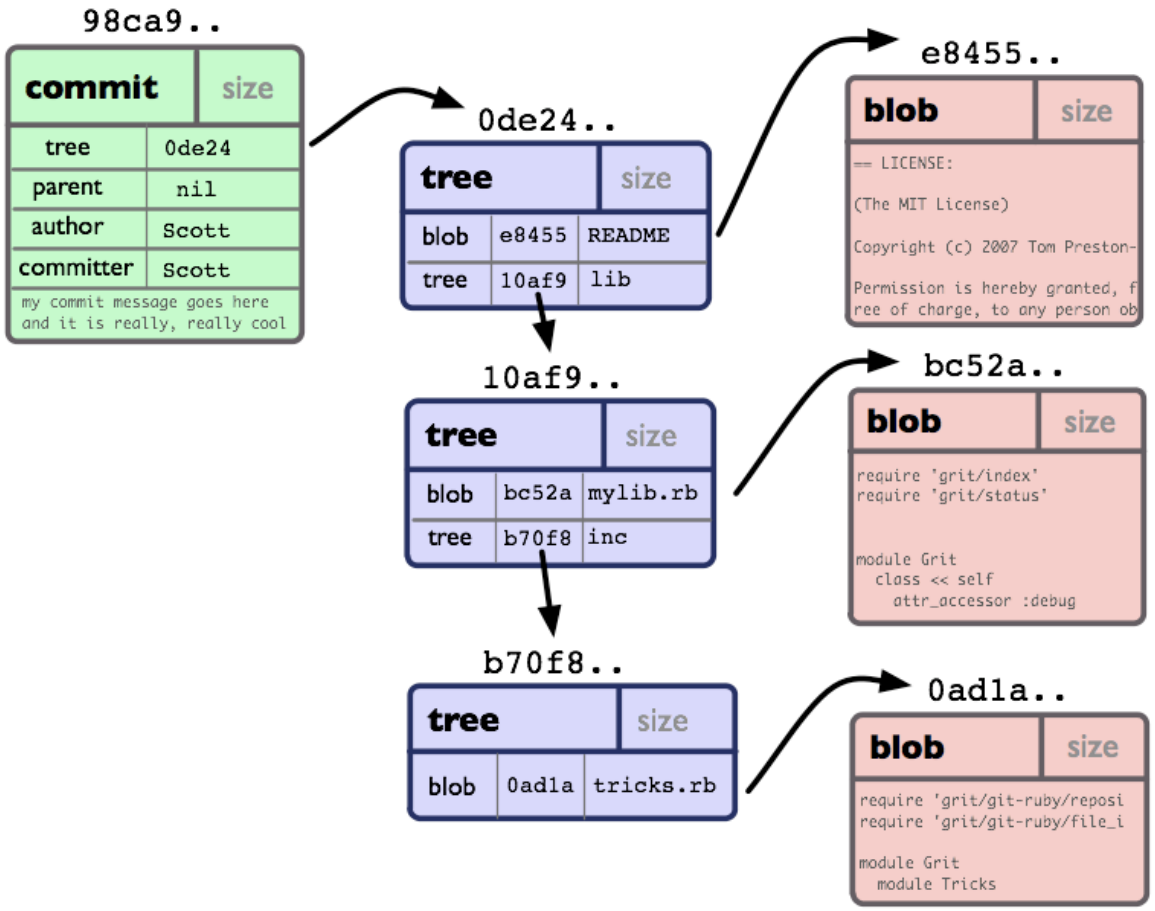
Git Nesneleri

4 tip nesne vardır:

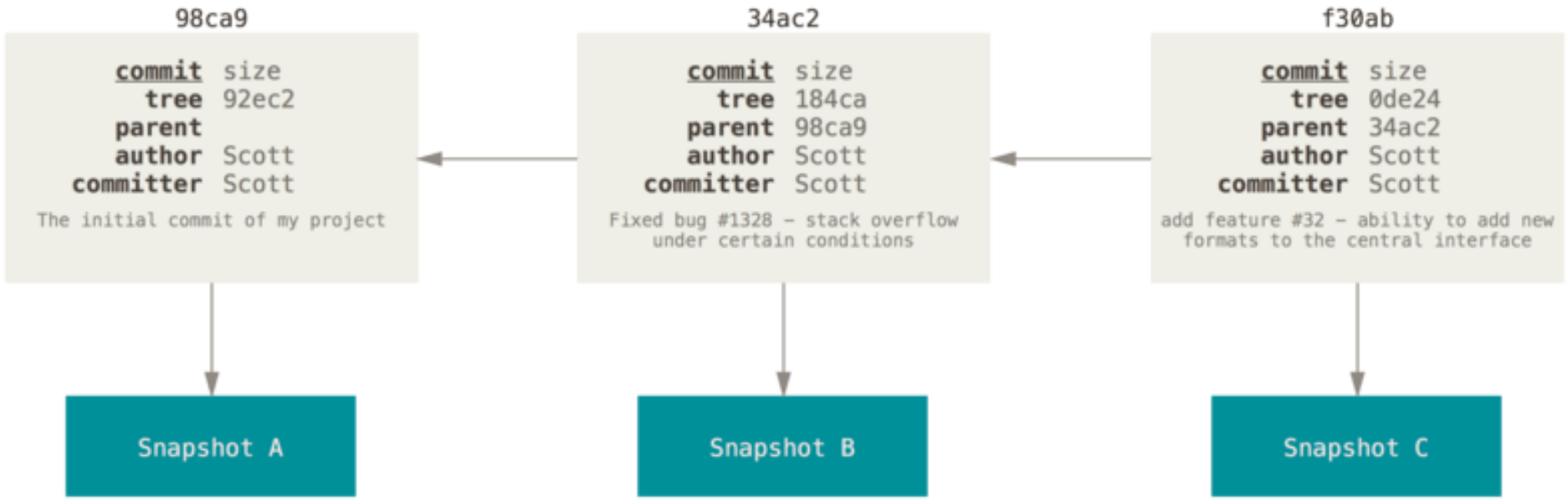
- **blob**: dosya verisi blob olarak tutulur.
- **tree**: Temel olarak dizin olarak düşünülebilir. Diğer alt dizinleri ve dosyaları adresler.
- **commit**: Bir tree nesnesine işaret eder. Belli bir zamanda projenin nasıl göründüğünü gösterir. Zaman bilgisi, değişiklikleri yapan kişi, bir önceki commit nesnesine referans gibi bilgileri içerir.
- **tag**: Commitleri etiketlemek için kullanılan bir yöntemdir. Örneğin sürüm ismi vermek gibi…



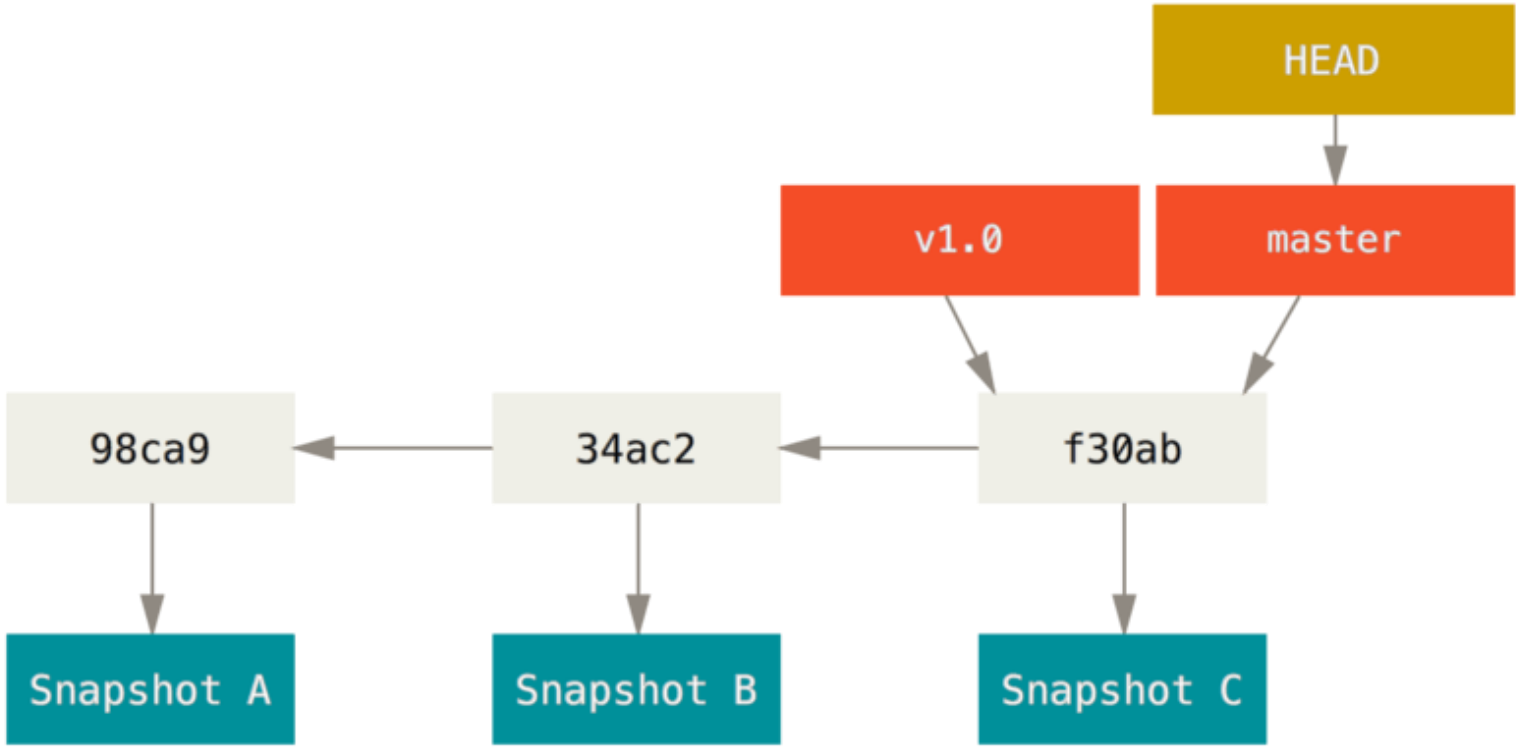
Git Nesneleri İlişkileri



Git Nesneleri



Git Nesneleri



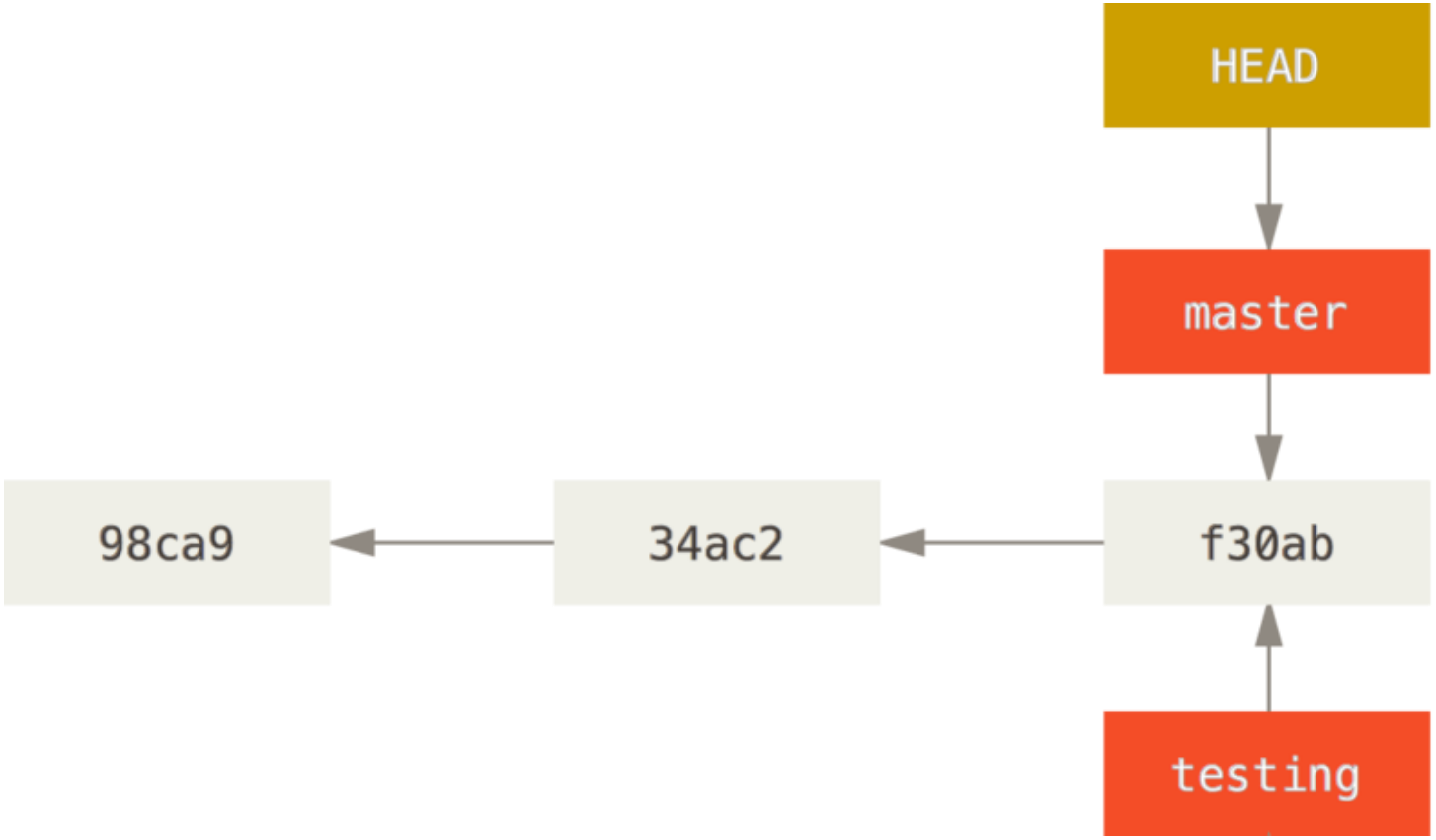
Dal (Branch)

Dal (branch) basit anlamda, bir commit'i gösteren hareketli bir işaretçiden başka bir şey değildir.

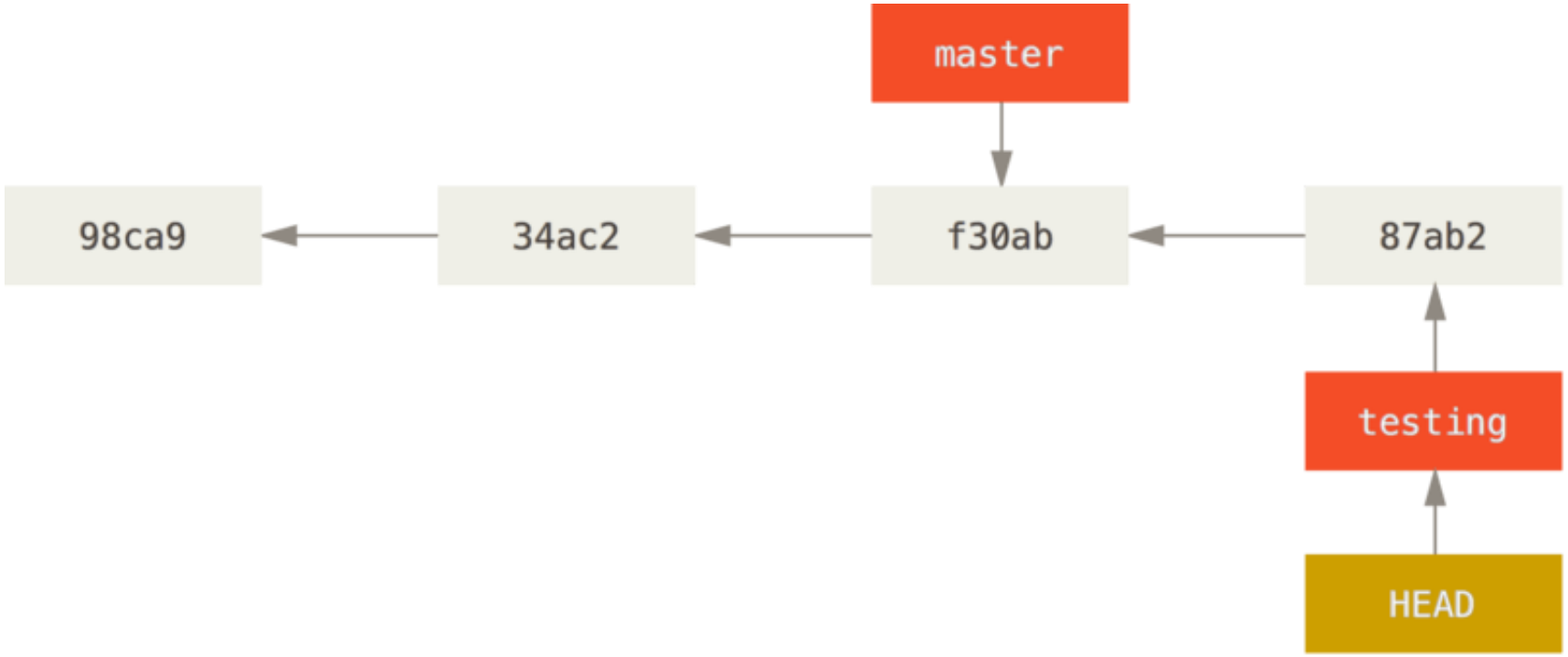
Varsayılan dal ismi, «master»'dır.

Her yapılan commit işleminde, mevcut dal yani hareketli işaretçi son commit'i gösterecek şekilde güncellenir.

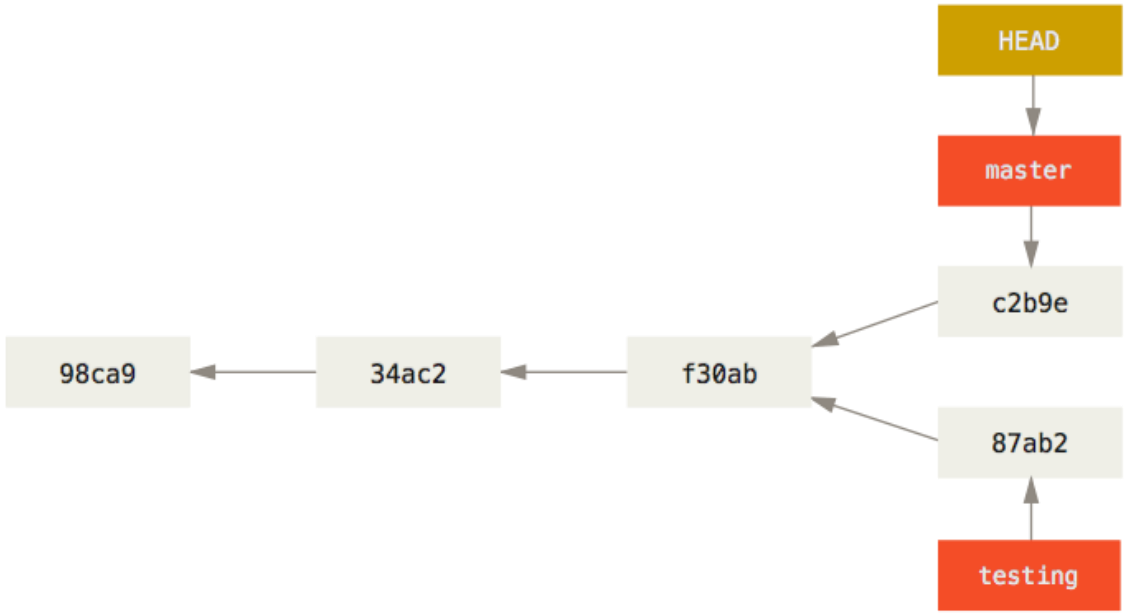
Dal



Dal



Dal



Dal

Komut satırında proje tarihçesini dallanlamalarla görmek için:

git log --oneline --decorate --graph --all

Git Deposu (Repository) Oluşturma

1. Mevcut dizini ilklendirme

cd /c/projects/git-sunumu

git init

2. Uzaktaki bir depoyu klonlama

git clone

Bu işlemlerden sonra .git dizini oluşur. İçeriğine bakalım…

.git dizininde ne var?

HEAD Mevcut dalı gösterir

config konfigürasyon verisi

description projenin açıklaması

index commit edilecek dosyalar bilgileri - binary bir dosyadır

logs/ ref'te yapılan değişikliklerin logları

objects/ tüm veri burada tutulur: commit, tree, blob ve tag nesneleri

hooks/ bir komuttan sonra çalıştırılacak kabuk betikleri bu dizinde tutulur

refs/ lokal ve uzak dal ve taglerin bilgilerini tutar

Git Deposu (Repository) Oluşturma / 2

Uzak bir git deposu çoklanarak (klonlanarak) da bir depo oluşturulabilir.

Bunun için klonlanacak deponun adresi gereklidir.

git clone https://github.com/zafergurel/git-sunumu.git <- Güvenli

git clone ssh://git@github.com/zafergurel/git-sunumu.git <- Güvenli, özel depolar için

git clone git://github.com/zafergurel/git-sunumu.git <- Halka açık depolar için, en hızlı yöntem

SSH, HTTPS ya da Git protokolü ile çoklama yapılabilir.

HTTPS için özel depolarda kullanıcı adı ve şifre gereklidir.

SSH kullanılırsa SSH anahtarı oluşturmak gereklidir.

SSH anahtar oluşturmak için Windows'ta PuttyGen kullanılabilir.

SSH'in herkese açık (public) anahtarı, deponun bulunduğu sunucuya tanımlanmalıdır.

Uygulamalar

1. Lokalde dosya ekleme

1. cd /c/projects/<proje adi>
2. git init
3. nano README
4. git status
5. git add
6. git status
7. git commit
8. git log

2. Lokalde dosya düzenleme

1. nano README
2. git status
3. git diff
4. git add .
5. git diff (Fark yok!)
6. git diff --staged (veya --cached – staging ortamındaki dosyalar için)
7. git commit –m «README düzenlendi»
8. git log -1 (son commiti gösterir)

3. Değişiklikleri geri alma

1. nano README (dosyayı değiştir ve kaydet)
2. git status -s (özet)
3. git checkout -- README (değiştirilmiş dosyanın üzerine son commit edileni yazar!)
4. git status -s

4. Değişiklikleri geri alma (stagingten)

1. nano README (dosyayı değiştir ve kaydet)
2. git add README
3. git status -s
4. git reset HEAD README (değiştirilen dosyayı değişikliklerin üzerine yazmadan çalışma alanına alır)
5. git add README (tekrar staging ortamına alalım)
6. git reset HEAD --hard (tüm değişiklikleri de geri alarak geri döndürür! Tüm dosyalarla beraber!)
7. git status

5. Aynı dosyanın iki hali

1. nano README (dosyayı değiştir ve kaydet)
2. git add README
3. git status –s
4. nano README (dosyayı tekrar değiştir ve kaydet)
5. git status (hem toplanma hem de çalışma alanında aynı dosya!)
6. git diff
7. git diff --staged
8. git add . (şimdi son değişiklikler dikkate alınır)
9. git commit -m «Bir değişiklik daha»
10. git log -1

6. Dosyaları Git Kontrolünden Çıkarma

1. touch app.dat
2. git status
3. echo *.dat > .gitignore (dat uzantılı dosyaları göndermeyelim)
4. git status

7. Dosya silme

1. touch sifreler.txt
2. git add sifreler.txt
3. git commit –m «sifreler»
4. git rm sifreler.txt
5. git status
6. git commit –m «sifreler silindi» (Gerçekten silindi mi? :D)

Silme ile ilgili Ara Bilgi

Bir dosya silinse bile Git tüm değişikliklerin tuttuğu için aslında veri kaybolmamaktadır.

Ama hassas bilgileri geçmişten temizlenin yolu var.

Bunun için git filter-branch komutu kullanılabilir. Bundan daha etkin olan diğer yöntem ise açık kaynak kodlu geliştirilmiş **BFG Repo Cleaner** aracı.

https://rtyley.github.io/bfg-repo-cleaner/

Araç kurulduktan sonra aşağıdaki komutlar ile proje geçmişi düzenlenir.

*java -jar bfg.jar --delete-files **sifreler.txt** my-repo.git*

git reflog expire --expire=now --all && git gc --prune=now --aggressive

8. Dosya yeniden isimlendirme

1. git mv README README.md
2. git status
3. git commit -m «isim degisikligi yapildi»

9. Son Commit'i Düzeltme

1. nano LICENSE
2. git add .
3. git commit -m «LICENSE eklendi»
4. git log --pretty=oneline -3 (son 3 değişiklik)
5. nano AUTHORS
6. git add .
7. git commit --amend –m «AUTHORS ve LICENSE eklendi»
8. git log --pretty=oneline -3

10. Logları inceleme

1. git log -2 (son iki log)
2. git log -2 –p (son iki log, değişikliklerle beraber --patched)
3. git log --pretty=format:"%h - %an, %ar : %s"
4. git log --pretty=oneline
5. git log --date=iso-strict (rfc ya da short da olabilir)
6. git log --since=3.day --before=1.day --pretty=oneline --author=zafer
7. git log --pretty=format:'%h %ad | %s%d [%an]' --graph --date=short
   git config --add alias.hist "log --pretty=format:'%h %ad | %s%d [%an]' --graph --date=short"
   git hist (Yukarıdaki log deyimi alias olarak tanımlanabilir)

11. Dal Oluşturma ve Birleştirme

1. git branch yeni-ozellik
2. git checkout yeni-ozellik
3. echo "alert(1);" > yeni-ozellik.js (yeni dosya ekle)
4. git add .
5. git commit -m "yeni dosya"
6. git log -1
7. git checkout master
8. git merge yeni-ozellik
9. git branch -d yeni-ozellik

12. Uzak Depoyu Klonlama

1. git clone ssh://git@192.168.56.101/root/git-sunumu.git
2. cd git-sunumu
3. touch README.md
4. git add README.md
5. git commit -m "README eklendi"
6. git push -u origin master

13. Uzak Depoya Kod Gönderme

1. nano AUTHORS
2. git add .
3. git commit -m "AUTHORS eklendi"
4. git push origin master

14. Uzak Depodan Kod Alma

1. git fetch (sadece uzak deponun commit bilgilerini alır)
2. git merge (uzaktaki dal ile mevcut dalı birleştirir)

Veya

git pull (git fetch & git merge)

Eğer hata çıkarsa git status ile çakışma olan dosyalar listelenir. Bu dosyalarda manuel işlem yapılarak düzeltme yapılır, dosyalar git add komutu ile eklenip git commit işlemi yapılır. Bu sayede merge işlemi tamamlanır.

15. Uzak Depodan Kod Alma / 2

Eğer çalışma dizininde değişmiş dosyalar varsa ilk önce bunlar stash komutu ile başka bir yerde kaydedilir ve son yapılan commite yani temiz çalışma dizinine dönülür.

1. git stash
2. git pull (değişiklikler güvenli şekilde çekilir)
3. git stash apply (saklanmış değişiklikler uzaktan gelen commitin üzerine uygulanır)

   Son işlemde dosya birleştirme hataları olursa bunlar manuel çözülür.

16. Uzak Depodaki Dalı Alma ve Silme

1. git fetch (uzak deponun lokal referanslarını günceller)
2. git checkout dal-ismi
3. git push origin --delete dal-ismi

17. Uzak Depoya Kod Göndereememe

Lokaldeki kodu uzak depoya göndermek her zaman mümkün olmayabilir.

Son yapılan git pull işleminden sonra başka bir kişi uzak depoyu güncellemiş olabilir.

Bu durumda, git push yapıldığında hata alınır.

1. git pull (İlk önce lokaldeki kod güncellenmeli.)
2. git status (Çatışma var mı? Varsa düzelt)
3. git add . (Tüm çatışmalar düzeltilip eklenir)
4. git commit -m "Yeni ozellik" (Yeni commit oluşturulur)
5. git push (Artık karşı tarafa gönderim yapılabilir)

18. Uzak Depo İşlemleri

1. git remote -v
2. git remote rename origin merkez
3. git remote show origin
4. git remote add yeni-remote <url>
5. git remote remove yeni-remote

19. Etiketleme

1. git tag v.1.0 (Basit etiketleme)
2. git tag v.1.0 -a -m "İlk sürüm" (Detaylı -annotated- etiketleme)
3. git tag -d v.1.0 (silme)
4. git tag (Listeleme)
5. git push --tags (Uzak depoya etiketleri gönderme)

Ekstralar

- SSH anahtarı ve Linux'a SSH ile erişim
- Açık kaynak kodlu Gitlab'i Ubuntu'ya Kurma

SSH Ayarları - SSH Anahtarı Oluşturma

Linux'a SSH ile uzaktan root ile bağlanmak için:

cd /root

mkdir .ssh

chmod .ssh 755

nano .ssh/authorized_keys && chmod 644 authorized_keys

Authorized Keys dosyasına, SSH public anahtarı yerleştirilir.

SSH Anahtarı Oluşturma

Linux ve Windows'ta Git Bash ile:
ssh-keygen -t rsa -C "<e-posta adresi>"

Windows:

Putty Key Generator (https://www.putty.org/)

Ubuntu'ya GitLab Kurulumu

Açık kaynak kodlu Gitlab'i Ubuntu üzerinde deneyebilirsiniz.

Windows'ta VirtualBox yazılımı ile Ubuntu kurabilir ve Gitlab'e uzaktan erişebilirsiniz.

https://www.digitalocean.com/community/tutorials/how-to-install-and-configure-gitlab-on-ubuntu-16-04

1. Gitlab kurulumu

```
sudo apt-get update
sudo apt-get install ca-certificates curl openssh-server postfix
cd /tmp
curl -LO https://packages.gitlab.com/install/repositories/gitlab/gitlab-ce/script.deb.sh
sudo bash /tmp/script.deb.sh
sudo apt-get install gitlab-ce
```

2. Firewall ayarları

```
sudo ufw status
sudo ufw allow http
sudo ufw allow https
sudo ufw allow OpenSSH
sudo ufw status
```

3. Gitlab'e Erişilicek Dış IP ya da DNS ismi konfigürasyona yazılmalı

```
sudo nano /etc/gitlab/gitlab.rb
sudo gitlab-ctl reconfigure
```

Kaynaklar

- Resimler, aşağıdaki adresten ücretsiz erişilebilen Pro Git kitabından alınmış (bazıları Türkçeleştirilmiş) ve kullanılmıştır.

https://git-scm.com/book/en/v2/Getting-Started-A-Short-History-of-Git