Übung 1: Grundlegender Repository-Workflow

Lernziele

Durch das Absolvieren dieser Übung werden Sie folgende Bereiche praktizieren:

- Initialisierung eines lokalen Git-Repositorys
- Grundlegender Git-Workflow: add, commit, push, pull
- Verfassen aussagekräftiger Commit-Nachrichten
- Verstehen der Beziehung zwischen lokalen und Remote-Repositorys

Szenario

Sie beginnen ein neues Datenanalyseprojekt und müssen Versionskontrolle für Ihre SQL-Skripte und Dokumentation einrichten. Sie werden ein Repository erstellen, einige anfängliche Dateien hinzufügen und den grundlegenden Git-Workflow praktizieren.

Setup

Initialisieren Sie ein leeres lokales Repository:

```
# Erstellen Sie ein neues Verzeichnis für Ihr Projekt
mkdir DatenanalyseProjekt
cd DatenanalyseProjekt

# Git-Repository initialisieren
git init

# Konfigurieren Sie Ihre Git-Identität (falls nicht bereits global eingestellt)
git config user.name "Ihr Name"
git config user.email "ihre.email@beispiel.com"
```

Teil 1: Erster Commit

1. Erstellen Sie eine `README.md`-Datei mit folgendem Inhalt:

```
# Datenanalyseprojekt

Dieses Projekt enthält SQL-Skripte zur Analyse von Kundendaten.

## Struktur
- `/scripts` - SQL-Skripte für die Datenanalyse
- `/docs` - Dokumentationsdateien
```

2. Erstellen Sie eine Verzeichnisstruktur:

```
mkdir scripts
mkdir docs
```

3. Erstellen Sie Ihr erstes SQL-Skript `scripts/kundenanalyse.sql`:

```sql
-- Kundenanalyse-Skript
-- Erstellt am: [Heutiges Datum]
-- Zweck: Analyse von Kundendemographie und Kaufmustern

SELECT
    kunden_id,
    vorname,
    nachname,
    registrierungsdatum,
    gesamtkaeufe
FROM kunden
WHERE registrierungsdatum >= '2024-01-01'
ORDER BY gesamtkaeufe DESC;
```

4. Überprüfen Sie den Status Ihres Repositorys und stagen Sie alle Dateien
5. Erstellen Sie Ihren ersten Commit mit der Nachricht: "Erster Commit: Projektstruktur und Kundenanalyse-Skript hinzugefügt"

Teil 2: Änderungen vornehmen

1. Ändern Sie das `kundenanalyse.sql`-Skript, indem Sie diese Abfrage am Ende hinzufügen:

```sql

-- Durchschnittlicher Kaufbetrag nach Kundensegment
SELECT
    kundensegment,
    AVG(kaufbetrag) as durchschnittskauf,
    COUNT(*) as kundenanzahl
FROM kunden k
JOIN kaeufe p ON k.kunden_id = p.kunden_id
GROUP BY kundensegment;
```

2. Erstellen Sie eine neue Datei `docs/analyse_notizen.md`:

```markdown

# Analyse-Notizen

## Wichtige Erkenntnisse
- [ ] Top 10 Kunden nach Kaufvolumen
- [ ] Kundensegmentierungs-Analyse
- [ ] Saisonale Kaufmuster
```

```
## Nächste Schritte
- [ ] Visualisierungs-Dashboard erstellen
- [ ] Automatisierte Berichterstattung einrichten
```

3. Stagen und committen Sie diese Änderungen mit der Nachricht: "Kundensegmentierungs-Abfrage und Analyse-Notizen hinzugefügt"

Teil 3: Übung für mehrere Commits

Erstellen Sie drei separate Commits für die folgenden Änderungen:

Commit 1: "Datenvalidierungs-Skript hinzugefügt" Erstellen Sie `scripts/datenvalidierung.sql`:

```
-- Datenvalidierungs-Skript
-- Überprüfung auf Datenqualitätsprobleme

-- Überprüfung auf doppelte Kundendatensätze
SELECT kunden_id, COUNT(*) as duplikatanzahl
FROM kunden
GROUP BY kunden_id
HAVING COUNT(*) > 1;

-- Überprüfung auf fehlende Pflichtfelder
SELECT COUNT(*) as fehlende_email_anzahl
FROM kunden
WHERE email IS NULL OR email = '';
```

Commit 2: "README mit neuen Skript-Informationen aktualisiert" Fügen Sie dies am Ende der `README.md` hinzu:

```
## Skripte
1. `kundenanalyse.sql` - Hauptabfragen für Kundenanalyse
2. `datenvalidierung.sql` - Datenqualitäts-Validierungsprüfungen
```

Commit 3: "Projekt-Zeitplan zur Dokumentation hinzugefügt" Erstellen Sie `docs/projekt_zeitplan.md`:

```
# Projekt-Zeitplan

## Phase 1: Datenerkundung (Woche 1)
- [x] Projekt-Repository einrichten
- [x] Erste Analyse-Skripte erstellen
- [ ] Daten-Profiling abschließen

## Phase 2: Analyse (Woche 2-3)
- [ ] Kundensegmentierungs-Analyse
- [ ] Kaufmuster-Analyse
```

```
- [ ] Zusammenfassende Berichte erstellen

## Phase 3: Berichterstattung (Woche 4)
- [ ] Dashboard erstellen
- [ ] Automatisierte Berichte erstellen
- [ ] Ergebnisse präsentieren
```

Teil 4: Überprüfung Ihrer Arbeit

1. Verwenden Sie `git log --oneline`, um Ihren Commit-Verlauf zu betrachten
2. Verwenden Sie `git status`, um zu überprüfen, dass Ihr Arbeitsverzeichnis sauber ist
3. Zählen Sie, wie viele Dateien Sie in Ihrem Repository haben
4. Verwenden Sie `git show HEAD`, um die Details Ihres letzten Commits zu sehen

Herausforderungen für erweiterte Übung

Herausforderung 1: Commit-Nachrichtenformate

Recherchieren Sie konventionelle Commit-Nachrichtenformate und schreiben Sie Ihre Commit-Nachrichten um, um dem Muster zu folgen:

- `feat:` für neue Features
- `docs:` für Dokumentationsänderungen
- `fix:` für Bugfixes

Erstellen Sie einen neuen Commit mit diesem Format.

Herausforderung 2: Mehrfaches Datei-Staging

1. Nehmen Sie Änderungen sowohl an `README.md` als auch an `kundenanalyse.sql` vor
2. Üben Sie das individuelle Stagen mit `git add <dateiname>`
3. Überprüfen Sie `git status` zwischen jedem add-Befehl
4. Committen Sie beide Änderungen zusammen

Herausforderung 3: Arbeitsverzeichnis-Untersuchung

1. Verwenden Sie `git ls-files`, um alle verfolgten Dateien zu sehen
2. Erstellen Sie eine temporäre Datei `temp.txt`, aber fügen Sie sie nicht zu Git hinzu
3. Verwenden Sie `git status`, um den Unterschied zwischen verfolgten und nicht verfolgten Dateien zu sehen
4. Löschen Sie die temporäre Datei

Erwartete Ergebnisse

Nach Abschluss dieser Übung sollten Sie haben:

- Ein Git-Repository mit 4-5 aussagekräftigen Commits
- Eine klare Verzeichnisstruktur mit Skripten und Dokumentation
- Erfahrung mit `git add`, `git commit`, `git status` und `git log`
- Verständnis für verfolgte vs. nicht verfolgte Dateien

- Übung im Schreiben beschreibender Commit-Nachrichten

Überprüfungs-Befehle

Führen Sie diese Befehle aus, um Ihre Arbeit zu überprüfen:

```
# Sollte 4-5 Commits zeigen
git log --oneline

# Sollte "nichts zu committen, Arbeitsbaum sauber" zeigen
git status

# Sollte alle Ihre verfolgten Dateien auflisten
git ls-files

# Sollte den Inhalt Ihres letzten Commits zeigen
git show HEAD --name-only
```

Diskussionsfragen

1. Warum ist es wichtig, aussagekräftige Commit-Nachrichten zu haben?
2. Wann sollten Sie einen neuen Commit erstellen vs. den vorherigen ändern?
3. Was ist der Unterschied zwischen dem Arbeitsverzeichnis und dem Git-Repository?
4. Wie hilft Ihnen `git status`, den aktuellen Zustand Ihres Projekts zu verstehen?

Hinweis: Diese Übung verwendet nur lokale Git-Operationen. In einem realen Szenario würden Sie auch ein Remote-Repository einrichten und `git push`- und `git pull`-Operationen üben.