Exercise 1: Basic Repository Workflow

Learning Objectives

By completing this exercise, you will practice:

- Initializing a local Git repository
- Basic Git workflow: add, commit, push, pull
- Writing meaningful commit messages
- Understanding the relationship between local and remote repositories

Scenario

You're starting a new data analysis project and need to set up version control for your SQL scripts and documentation. You'll create a repository, add some initial files, and practice the basic Git workflow.

Setup

Initialize an empty local repository:

```
# Create a new directory for your project
mkdir DataAnalysisProject
cd DataAnalysisProject

# Initialize Git repository
git init

# Configure your Git identity (if not already done globally)
git config user.name "Your Name"
git config user.email "your.email@example.com"
```

Part 1: First Commit

1. Create a `README.md` file with the following content:

```
# Data Analysis Project

This project contains SQL scripts for analyzing customer data.

## Structure
- `/scripts` - SQL scripts for data analysis
- `/docs` - Documentation files
```

2. Create a directory structure:

```
mkdir scripts
mkdir docs
```

3. Create your first SQL script `scripts/customer_analysis.sql`:

```sql
-- Customer Analysis Script
-- Created: [Today's Date]
-- Purpose: Analyze customer demographics and purchase patterns

SELECT
    customer_id,
    first_name,
    last_name,
    registration_date,
    total_purchases
FROM customers
WHERE registration_date >= '2024-01-01'
ORDER BY total_purchases DESC;
```

4. Check the status of your repository and stage all files
5. Create your first commit with the message: "Initial commit: Add project structure and customer analysis script"

Part 2: Making Changes

1. Modify the `customer_analysis.sql` script by adding this query at the end:

```sql

-- Average purchase amount by customer segment
SELECT
    customer_segment,
    AVG(purchase_amount) as avg_purchase,
    COUNT(*) as customer_count
FROM customers c
JOIN purchases p ON c.customer_id = p.customer_id
GROUP BY customer_segment;
```

2. Create a new file `docs/analysis_notes.md`:

```markdown

# Analysis Notes

## Key Findings
- [ ] Top 10 customers by purchase volume
- [ ] Customer segmentation analysis
- [ ] Seasonal purchase patterns
```

```
## Next Steps
- [ ] Create visualization dashboard
- [ ] Set up automated reporting
```

3. Stage and commit these changes with the message: "Add customer segmentation query and analysis notes"

Part 3: Multiple Commits Practice

Create three separate commits for the following changes:

Commit 1: "Add data validation script" Create `scripts/data_validation.sql`:

```
-- Data Validation Script
-- Check for data quality issues

-- Check for duplicate customer records
SELECT customer_id, COUNT(*) as duplicate_count
FROM customers
GROUP BY customer_id
HAVING COUNT(*) > 1;

-- Check for missing required fields
SELECT COUNT(*) as missing_email_count
FROM customers
WHERE email IS NULL OR email = '';
```

Commit 2: "Update README with new script information" Add this to the end of `README.md`:

```
## Scripts
1. `customer_analysis.sql` - Main customer analysis queries
2. `data_validation.sql` - Data quality validation checks
```

Commit 3: "Add project timeline to documentation" Create `docs/project_timeline.md`:

```
# Project Timeline

## Phase 1: Data Exploration (Week 1)
- [x] Set up project repository
- [x] Create initial analysis scripts
- [ ] Complete data profiling

## Phase 2: Analysis (Week 2-3)
- [ ] Customer segmentation analysis
- [ ] Purchase pattern analysis
- [ ] Create summary reports
```

```
## Phase 3: Reporting (Week 4)
- [ ] Build dashboard
- [ ] Create automated reports
- [ ] Present findings
```

Part 4: Review Your Work

1. Use `git log --oneline` to view your commit history
2. Use `git status` to verify your working directory is clean
3. Count how many files you have in your repository
4. Use `git show HEAD` to see the details of your latest commit

Challenges for Advanced Practice

Challenge 1: Commit Message Formats

Research conventional commit message formats and rewrite your commit messages to follow the pattern:

- `feat:` for new features
- `docs:` for documentation changes
- `fix:` for bug fixes

Create a new commit using this format.

Challenge 2: Multiple File Staging

1. Make changes to both `README.md` and `customer_analysis.sql`
2. Practice staging them individually with `git add <filename>`
3. Check `git status` between each add command
4. Commit both changes together

Challenge 3: Working Directory Investigation

1. Use `git ls-files` to see all tracked files
2. Create a temporary file `temp.txt` but don't add it to Git
3. Use `git status` to see the difference between tracked and untracked files
4. Delete the temporary file

Expected Outcomes

After completing this exercise, you should have:

- A Git repository with 4-5 meaningful commits
- A clear directory structure with scripts and documentation
- Experience with `git add`, `git commit`, `git status`, and `git log`
- Understanding of tracked vs untracked files
- Practice writing descriptive commit messages

Verification Commands

Run these commands to verify your work:

```
# Should show 4-5 commits
git log --oneline

# Should show "nothing to commit, working tree clean"
git status

# Should list all your tracked files
git ls-files

# Should show the content of your last commit
git show HEAD --name-only
```

Discussion Questions

1. Why is it important to have meaningful commit messages?
2. When should you create a new commit vs. modifying the previous one?
3. What's the difference between the working directory and the Git repository?
4. How does `git status` help you understand the current state of your project?

Note: This exercise uses only local Git operations. In a real scenario, you would also set up a remote repository and practice `git push` and `git pull` operations.