**Nachdenkzettel Git:  
  
Aufgabe 1:**

* Versionskontrolle: Ermöglicht die Nachverfolgung von Änderungen im Code über die Zeit.
* Zusammenarbeit: Ermöglicht effiziente Zusammenarbeit zwischen mehreren Entwicklern.
* Rückverfolgbarkeit: Dokumentiert, wer welche Änderungen gemacht hat.
* Branching und Merging: Ermöglicht das parallele Arbeiten an verschiedenen Features und das Zusammenführen von Codeänderungen.
* Sicherung: Schützt vor Datenverlust durch regelmäßige Backups des Codes.

**Aufgabe 2:**

* .java: Ja, da dies Quellcode für Java-Programme ist.
* .xml: Ja, oft für Konfigurationsdateien oder Datenaustauschformate verwendet.
* .json: Ja, ebenfalls für den Datenaustausch und Konfigurationen.
* Bilddateien: Ja, wenn sie im Projekt benötigt werden (z. B. für Benutzeroberflächen).
* Musikdateien: Nein, es sei denn, sie sind Teil des Projekts (z. B. Soundeffekte in einer Anwendung).
* UML Modelle: Ja, wenn sie zur Dokumentation oder Visualisierung des Projekts dienen.
* Notizen und Dokumentationen: Ja, um Projektinformationen zu speichern.
* Vertrauliche Daten: Nein, solche Daten sollten nicht im Repository gespeichert werden.
* Abschlussarbeiten: Nein, persönliche Dokumente gehören nicht ins Repository.
* Log-Files: Nein, sie sollten lokal erzeugt und ignoriert werden.
* Konfigurationen: Ja, wenn sie zur Konfiguration des Projekts benötigt werden.

**Aufgabe 3 (in IntelliJ):**

**1. Hinzufügen zur Staging-Area:**

Wenn Sie Änderungen an Ihren Dateien vorgenommen haben und diese für einen Commit vorbereiten möchten, fügen Sie diese der Staging-Area hinzu:

1. Öffnen Sie ein Git-fähiges Projekt in Ihrer Entwicklungsumgebung (z.B. IntelliJ IDEA).
2. Überprüfen Sie die unversionierten Änderungen im Fenster "Changes" oder "Version Control".
3. Wählen Sie die Dateien aus, die Sie hinzufügen möchten, und fügen Sie sie zur Staging-Area hinzu.

**2. Commit:**

Nach dem Hinzufügen von Änderungen zum Staging-Bereich wird ein Commit erstellt, um die Änderungen dauerhaft zu speichern:

1. Geben Sie eine aussagekräftige Commit-Nachricht ein, die die durchgeführten Änderungen beschreibt.
2. Führen Sie den Commit durch, um die Änderungen lokal zu speichern.

**3. Push:**

Um Ihre lokalen Commits mit dem Remote-Repository zu synchronisieren, führen Sie einen Push durch:

1. Klicken Sie auf die Schaltfläche "Push" oder wählen Sie die Option im Menü Ihrer Entwicklungsumgebung.
2. Geben Sie gegebenenfalls die erforderlichen Anmeldedaten für das Remote-Repository ein.

**4. Pull:**

Um Änderungen aus dem Remote-Repository zu erhalten und Ihr lokales Repository zu aktualisieren, führen Sie einen Pull durch:

1. Klicken Sie auf die Schaltfläche "Pull" oder wählen Sie die Option im Menü Ihrer Entwicklungsumgebung.
2. Geben Sie wieder gegebenenfalls die erforderlichen Anmeldedaten für das Remote-Repository ein.

**Aufgabe 4:**

1. Konflikte erkennen: Git informiert den Entwickler über einen Merge-Konflikt, wenn Änderungen in verschiedenen Branches miteinander konkurrieren.
2. Manuelle Anpassung: Der Entwickler muss die Konfliktstellen im Code manuell beheben, indem er entscheidet, welche Änderungen beibehalten werden sollen.
3. Hinzufügen und Commit: Nachdem der Konflikt gelöst wurde, muss der Entwickler die geänderten Dateien zur Staging-Area hinzufügen und einen Commit erzeugen.

**Aufgabe 5:**

Push-Zeitpunkt: Entwickler sollten regelmäßig pushen, aber nicht unvollständige oder nicht funktionierende Änderungen.

Veröffentlichungsfrequenz: Veröffentlichen Sie Änderungen häufig, insbesondere nach der Fertigstellung von Features oder Bugfixes.

Commit-Messages: Verwenden Sie aussagekräftige Commit-Messages, die den Zweck der Änderungen klar beschreiben. Eine gute Commit-Message hilft anderen Entwicklern, den Kontext und den Grund für die Änderungen zu verstehen.