**Neue Repository erstellen und hochladen:**

1. echo "# Test1" >> README.md:
   * Fügt den Text "# Test1" zur Datei README.md hinzu oder erstellt die Datei, falls sie noch nicht existiert.
2. git init:
   * Initialisiert ein neues Git-Repository im aktuellen Verzeichnis.
3. git add README.md (oder einfach am Ende „.“ Setzen und alle Dateien werden hinzugefügt):
   * Fügt die README.md-Datei dem sogenannten "Staging-Bereich" hinzu, um Änderungen für den nächsten Commit vorzubereiten.
4. git commit -m "first commit":
   * Nimmt alle Änderungen im Staging-Bereich und erstellt einen neuen Commit mit der angegebenen Commit-Nachricht ("first commit").
5. git branch -M main:
   * Benennt den Standard-Branch von "master" in "main" um. Dieser Befehl ist optional und hängt von den Einstellungen auf GitHub ab.
6. git remote add origin https://github.com/Etherial96/Test.git:
   * Verbindet das lokale Repository mit dem GitHub-Repository unter dem Namen "origin". Es kann auch anders als „origin“ genannt werden.
7. git push -u origin main:
   * Lädt die lokalen Commits auf GitHub hoch und verbindet den lokalen "main"-Branch mit dem "origin"-Repository. Der Parameter "-u" setzt den "upstream" und ermöglicht zukünftige "git push"-Befehle ohne zusätzliche Angaben.

**Bestehendes Repository hochladen:**

1. git remote add origin https://github.com/Etherial96/Test1.git:
   * Verbindet das lokale Repository mit dem bereits bestehenden GitHub-Repository unter dem Namen "origin".
2. git branch -M main:
   * Benennt den Standard-Branch von "master" in "main" um. Dieser Befehl ist optional und hängt von den Einstellungen auf GitHub ab.
3. git push -u origin main:
   * Lädt die lokalen Commits auf GitHub hoch und verbindet den lokalen "main"-Branch mit dem "origin"-Repository. Der Parameter "-u" setzt den "upstream" und ermöglicht zukünftige "git push"-Befehle ohne zusätzliche Angaben.

**Zusatz:**

1. git clone <https://github.com/username/mein-projekt.git> : Clone damit das Projekt auf dem lokalen Rechner
2. git status : gibt den status an von den dateien auf dem lokalen Rechner
3. git config --global user.email “[beispiel@hotmail.de](mailto:beispiel@hotmail.de)” : erstellt lokal auf den rechner das Profil
4. git config –global user.name “Dein Name”
5. git branch feature-newStuff : man kann auch den branch wechseln innerhalb eines branch
6. git checkout feature-anmeldung: man verlässt den branch in dem man sich aufhält und geht in den Branch „feature-anmeldung.
7. git merge feature-anmeldung : Sagen wir ich bin der main und führe den Befehl aus, dann werden die Dateien/Änderungen in der main geupdated.
8. git push origin --delete zu-loeschender-branch : löscht das online repository Branch
9. git pull origin gewuenschter-branch : download des Branch

Begriffserklärungen:

„remote repository“ befindet sich auf meinem Git-Profil.

Branch: Es gibt Bspw. Eine Ordner Struktur im remote Repository. Diese kann man runterladen und auch abändern. Es ist wie eine Version 1.0 der man aber den Namen bsp. „FirstOne“ gibt. Möchte man diese Hochladen ist diese quasi in der selben Branch wieder „FirstOne“. Es kann sinnvoll sein nach dem herunterladen einen neuen branch zu erstellen „SecondOne“ und dann zu pushen. So hat man eine komplett neue Strukur in seinem remote repository. Die Dateien könnten komplett andere sein und nichts mit dem Projekt zu tun haben, aber wäre nicht sinnvoll.

Beispiel: Teamchef ändert immer die HauptBranch ab. Zwei andere Entwickler haben einen Teilbereich des Projekts und pushen ihre eigene Branch. Der Teamchef kann darauf zu greifen und alles zusammenfügen im HauptBranch.

Merge ist, wenn man die lokale Datei fertig bearbeitet hat und danach die Dateien in der Branch updated.

In größeren Entwicklungsteams ist es oft nicht praktikabel, alle Entwickler individuell zu benachrichtigen, wenn an einer bestimmten Funktion gearbeitet wird. Stattdessen gibt es gängige Praktiken und Werkzeuge, die die Kommunikation und Zusammenarbeit im Team erleichtern:

1. **Kommunikationsplattformen:**
   * Nutze Team-Kommunikationsplattformen wie Slack, Microsoft Teams oder ähnliche Tools. Erstelle Channels oder Chatgruppen für verschiedene Themen oder Funktionen, und teile dort Informationen über deine Arbeit mit.
2. **Projektmanagement-Tools:**
   * Verwende Projektmanagement-Tools wie Jira, Trello oder GitHub Issues. Erstelle Aufgaben oder Tickets für neue Funktionen, und weise sie den betreffenden Entwicklern zu. Teammitglieder können Benachrichtigungen erhalten, wenn es Updates oder Kommentare zu den Aufgaben gibt.
3. **Pull Requests oder Merge-Anfragen:**
   * Wenn das Team Git-Repositories und Pull Requests (GitHub, GitLab, Bitbucket) verwendet, können Entwickler Benachrichtigungen erhalten, wenn neue Branches erstellt oder Pull Requests eröffnet werden. Dies kann als indirekte Benachrichtigung dienen.
4. **Regelmäßige Meetings oder Standups:**
   * Halte regelmäßige Meetings oder Standups ab, in denen Entwickler über ihre aktuellen Arbeiten sprechen können. Dies bietet eine Gelegenheit, Informationen zu teilen und Feedback zu erhalten.
5. **Automatisierte Benachrichtigungen:**
   * Einige Entwicklungsumgebungen und CI/CD-Systeme können automatisierte Benachrichtigungen senden, wenn bestimmte Aktionen oder Ereignisse auftreten. Dies kann nützlich sein, um Teammitglieder über wichtige Änderungen zu informieren.

Die genaue Vorgehensweise hängt von den Präferenzen und den Tools ab, die dein Team verwendet. Es ist wichtig, transparent zu sein und sicherzustellen, dass es klare Kommunikationskanäle gibt, um sicherzustellen, dass Teammitglieder auf dem Laufenden sind und effizient zusammenarbeiten können.