Repository erstellen

Öffne Git Bash um git zu starten

|  |  |
| --- | --- |
| pwd | Zeigt aktuelles Verzeichnis an |
| cd .. | Gehe Verzeichnis hoch |
| cd c/users/anpa02… | Navigiere in ausgewähltes Verzeichnis (oder git bash direkt im Ordner öffnen) |
| git init | Lege repository in gewähltem Ordner an |
| git add \*dateiname\* | Bereite Datei zum Commiten vor (Stagen) |
| git commit –m „Kommentar“ | Commite Datei (git add muss vorher ausgeführt werden)  -m: Ist die Message, die man hinzufügt |
| Git log | Zeigt der Versionslog an |

Git Änderungen vornehmen

|  |  |
| --- | --- |
| z.B. html-Datei:  nano index.html   * Text einfügen * Mit „Strg“ durch Menü (siehe unten navigieren) * Z.b. strg + s (speichern) * Strg + x (beenden) |  |
| Git add . | Stage alle Dateien eines Projekts |
| Git add --all | Stage alle Änderungen |

Git Änderungen prüfen

|  |  |
| --- | --- |
| git status | Überprüfe Status der Datein |

Git Änderungen Commiten

|  |  |
| --- | --- |
| git commit –m | Alles wird commitet (hinter –m Kommentar) |

* Master = aktuelle Version (Head Version)

Git Versionen vergleichen

|  |  |
| --- | --- |
| Git diff \*dateiname\* | In Rot ursprüngliche Version, in Grün die neue |
| Allternativ über tortoise:  Rechtsklick auf Datei 🡪 tortoise 🡪 Diff | Zeigt in einem gesplitteten Fenster die alte und die neue Version mit Unterschieden |

Git Änderungen verwerfen

|  |  |
| --- | --- |
| Git reset –hard HEAD | „Harte“ Rückkehr zum zuletzt comitteten Version |

Git Entwicklungszweige anlegen

|  |  |
| --- | --- |
| Git log | Listet Versionslog auf |
| Git checkout \*hash Nummer\*🡪 hash nummer muss eindeutig sein, u.U. reichen die ersten 5 Ziffern | Wähle Version aus und wiederherstelle sie |
| **Git checkout --b** „Branchname“ | Lege neuen Branch (Verzweigung) an |
| Git log --all | Zeige alle Versionen an |
| Git log --all --graph | Zeige Branch-Hierarchie an |
| Git log --all --graph --oneline | Wie drüber, nur kompakt dargestellt |
| **Git checkout** \*brach Name\* | Wechseln zwischen Branches |

Git Dateien löschen/entfernen

|  |  |
| --- | --- |
| Lösche die Datei manuell 🡪  Git add \*dateiname\*  Git commit –m „…“ | Manuelles Löschen |
| Git rm \*dateiname\*  Git status  Git commit –m „…“ | Vorbereitung des Löschen, beinhaltet bereits das adden  Zeigt Status der Datei an |

Git Entwicklungszweige zusammenführen

|  |  |
| --- | --- |
| Git checkout –b \*brachname\*  Git checkout master (oder main)  Git merge \*branchname\* | Lege Branch an  Wechsel in den master-Branch  Führe master und branch zusammen |
| Git checkout \*branchname\* | Wechsel in eigenen Branch |
| U.U. könnte Fenster mit Doku sich öffnen. Hier dokumentiert man, warum Merge notwendig ist.  :q quit der Merge-Beschriebung (mit enter bestätigen) |  |

Fallbeispiel: Anlegen eines eigenen Branches lokal + merge mit main

|  |  |
| --- | --- |
| Git checkout –b \*brachname\* | Lege Branch an + Nehme Änderungen vor |
| Git commit –m “Kommentar” | Commite Änderungen |
| Git branch –a  Git push --set-upstream origin \*branchname\* | Ordne aktuellen Branch dem main repository zu |
| Git checkout main | Wechsele zum main Repository |
| Git merge \*branchname\* | Mergen des neuen Branches mit main Repository |

Fallbeispiel: Branch aus Github ziehen, ändern und mergen

|  |  |
| --- | --- |
| Mit tortoise Branch lokal installieren  Änderung vornehmen |  |
| Git add .  Git commit –m „Kommentar“  Git push | Branch aktuelisieren |
| Git checkout main | Rückkehr zur main repository |
| Git merge \*branchname\* | Es öffnet sich ein Fenster, in dem man den merge kommentieren kann.  Mit :q! beenden |
| Git push | Änderungen werden im main gespeichert |

Rollen

|  |  |
| --- | --- |
| Guest | Lese- und Schreibzugirff auf den Issue Tracker, Lesezugriff auf das Wiki. Kein Zugriff auf Dateien im Repository! |
| Reporter | Wie Gast aber mit Lesezugriff auf Dateien im Repository |
| Developer | Wie Reporter aber mit Schreibzugriff auf das Repository |
| Maintainer | Wie Developer aber mit Zugriff auf Projekteinstellungen |
| Owner | Zugriff auf Projekteinstellungen |

<https://git.rwth-aachen.de/help/user/permissions>

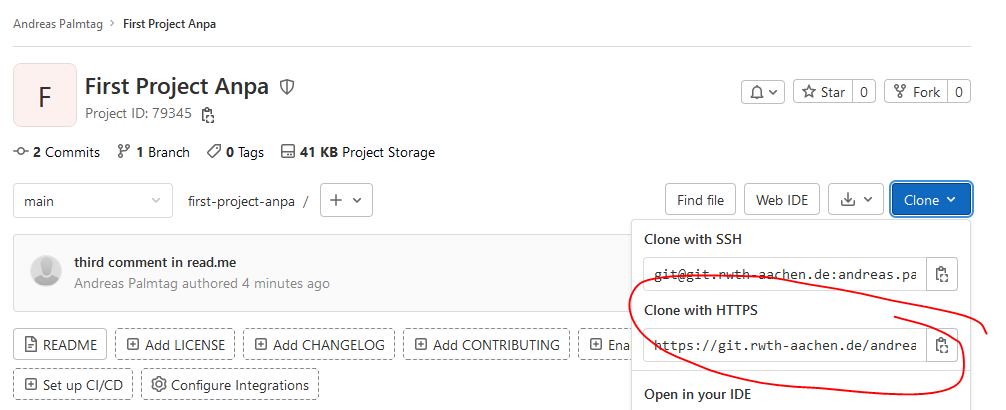
Repository

|  |  |
| --- | --- |
| Push to a remote repository | Replizieren Sie Änderungen an diesem Repository in ein anderes Repository |
| Protected Branches und Protected Tags | Schützen Sie Entwicklungszweige vor versehentlichen Änderungen |
| Deploy Keys und Deploy Token | Richten Sie Maschinenaccounts ein, die auf das Repository zugreifen können. |

Achtung: Standardmäßig ist der Entwicklungszweig master als Protected Branch konfiguriert, sodass keine Änderungen gemacht werden können. Dies sollten Sie hier ändern.

Projekt erstellen

1. Projekt auf Gitlab anlegen und URL kopieren



1. Lokale Kopie erstellen

|  |  |
| --- | --- |
| Rechtsklick Git Bash öffnen | Kommandozeile öffnen |
| Mkdir \*ordnername\* | Erstelle Ordner |
| Cd \*ordnername\* | Wechsele in Ordner rein |
| Git clone \*Projekt-URL\* | Siehe obene (HTTPS!) |

1. Änderungen comitten + In main-branch überschreiben

|  |  |
| --- | --- |
| Git add --all  Git commit –m „Kommentar“ | Geänderte Datei commiten |
| Git push | Änderungen werden auf Server geladen |

Formatierung eins Wikis

<https://git.rwth-aachen.de/help/user/markdown#code-spans-and-blocks>

Erfahrung

* Projekt auf Gitlab löschen und lokal nicht
  + Durch push des lokalen Projektes kann es auf Gitlab wieder hergestellt werden
* Gitignore-File 🡪 entferne einzelne subdirectories aus commit
* Sollte an verschiedenen stellen an einem Branch gearbeitet werden, kann über git pull aktualisiert werden
* Branch lokal löschen: git brnach –D \*branchname\*