Häufig genutzte Git-Kommandos auf einen Blick

Git Cheat Sheet

Tim Schürmann

Einrichten und Konfigurieren	
Benutzername und E-Mail-Adresse müssen Sie vorab hinterlegen. Dabei handelt es sich nicht um die Zugangsdaten zu einem Dienst wie Github. Die Option `--global` legt die Einstellung für alle (lokalen) Git-Repositories fest, `--local` setzt die Einstellungen für das aktuelle Repository.	
`git config --global user.name "Eike Pinguin"`	Hinterlegt den Benutzernamen.
`git config --global user.email "eike@example.com"`	Hinterlegt die E-Mail-Adresse.
`git config --global core.editor "nano"`	Legt den Texteditor fest (hier Nano).
`git help --config`	Listet alle möglichen Einstellungen auf. Beschreibungen dazu liefert `git help config`.
`git config --list`	Listet alle gesetzten Einstellungen auf.
`git help config`	Ruft Hilfe zum angegebenen Git-Befehl auf (hier `config`).
Repository erstellen oder klonen	
Ein Repository dient als Speicherort für alle zu verwaltenden Projektdateien wie Quellcode, Konfigurationsdateien und so weiter. Ein lokales Repository besteht aus einem Verzeichnis. Ein Remote Repository offeriert in der Regel ein Server oder ein Dienst wie Github oder GitLab.	
`git init`	Erstellt im aktuellen Verzeichnis ein Repository.
`git clone ~/original`	Klont das Repository im Verzeichnis `~/original` in das aktuelle Verzeichnis.
`git clone https://example.com/path/to/repo.git`	Klont das angegebene Remote Repository via HTTPS in das aktuelle Verzeichnis.
`git clone ssh://user@example.com:/path/to/repo.git`	Klont das angegebene Remote Repository via SSH in das aktuelle Verzeichnis.
`git clone git://example.com:/path/to/repo.git`	Klont das angegebene Remote Repository via Git-Protokoll in das aktuelle Verzeichnis.
`git clone --branch foo https://example.com/user/repo.git`	Klont nur den Branch namens `foo`.
Staging Area – Dateien zur Versionierung vormerken	
Modifizierte und neu erstellte Dateien landen zunächst in der Staging Area. Nur die dort gesammelten Projektänderungen bilden später einen neuen Zwischenstand (Commit), zu dem Sie zurückkehren können. Git ignoriert dabei alle Dateien und Verzeichnisse, die die Datei `.gitignore` auflistet.	
`git add main.c`	Schiebt den aktuellen Zustand der Datei `main.c` in die Staging Area.
`git add .`	Steckt alle Dateien und Verzeichnisse in die Staging Area.
`git add src`	Steckt das Verzeichnis `src/` in die Staging Area.
`git rm main.c`	Löscht die Datei `main.c` und notiert das in der Staging Area.
`git mv main.c src`	Verschiebt die Datei `main.c` in das Verzeichnis `src` und notiert das in der Staging Area.
`git mv main.c func1.c`	Benennt die Datei `main.c` in `func1.c` um und notiert das in der Staging Area.
`git reset`	Leert die Staging Area.
`git reset main.c`	Entfernt die Datei `main.c` wieder aus der Staging Area.
`git diff main.c`	Vergleicht die Datei `main.c` mit deren Stand in der Staging Area.
`git diff`	Vergleicht alle Dateien mit ihrem jeweiligen Stand in der Staging Area.
`git status`	Aktueller Status der Staging Area.
Commits – Schnappschüsse erstellen	
Ein Commit ist ein Zwischenstand des Projekts, zu dem Sie jederzeit zurückkehren können. Jeder Commit erhält eine eindeutige Identifikationsnummer, die Commit ID. Einem Commit heften Sie Tags und einen Kommentar (Commit Messages) an.	
`git commit`	Erstellt einen neuen Commit mit allen in der Staging Area gesammelten Änderungen.
`git commit -m "Commit Message"`	Heftet dem Commit direkt die Nachricht `Commit Message` an.
`git commit -am "Commit Message"`	Überträgt alle geänderten Dateien in den Staging-Bereich und erstellt direkt einen neuen Commit mit der Nachricht `Commit Message`.
`git commit --amend`	Ändert die letzte Commit-Nachricht (dabei entsteht ein neuer Commit mit neuer ID).
`git tag -a version3.12 12345`	Heftet dem Commit mit der ID `12345` das Tag `version3.12` an.
`git tag`	Listet alle bereits vergebenen Tags auf.
Commit Log – Commits unter der Lupe	
`git log`	Listet alle Commits auf (für den aktiven Branch).
`git log -3`	Zeigt die letzten drei Commits.
`git log --stat`	Liefert alle Commits und die dabei jeweils umgesetzten Änderungen.
`git log --oneline`	Fasst jeden Commit in einer Zeile zusammen.
`git log --after="2024-04-22"`	Liefert alle Commits, die nach dem 22. April 2024 erfolgt sind.
`git log --before="2024-04-22"`	Liefert alle Commits, die vor dem 22. April 2024 erfolgt sind.
`git log -p main.c`	Zeigt die Veränderungen der Datei `main.c` im Laufe der Zeit an.
`git log --grep="patch"`	Sucht in allen Commit Messages nach dem Text `patch`.
`git diff 12345 67890`	Ermittelt die Unterschiede zwischen den zwei Commits mit den IDs `12345` und `67890`.

Häufig genutzte Git-Kommandos auf einen Blick (Seite 2)

Git Cheat Sheet

Tim Schürmann

Commit Log – Commits unter der Lupe	
`git show 12345`	Liefert alle Informationen über den Commit mit der ID 12345.
`git show 12345 main.c`	Verrät alle Veränderungen in der Datei `main.c` beim Commit mit der ID 12345.
Änderungen zurücknehmen	
HEAD steht stellvertretend für den letzten Commit, HEAD~*n* ist die Abkürzung für den *n*-ten Vorgänger. HEAD~2 bezeichnet beispielsweise den vorletzten Commit.	
`git checkout HEAD main.c`	Setzt die Datei `main.c` auf den Stand beim letzten Commit zurück.
`git reset --hard HEAD~1`	Nimmt den letzten Commit zurück.
`git reset HEAD~1`	Nimmt den letzten Commit zurück, ändert aber nicht die Dateien im Projektverzeichnis.
`git reset --hard 12345`	Kehrt zurück zum Commit mit der ID 12345.
`git reset 12345`	Kehrt zurück zum Commit mit der ID 12345, ändert aber nicht die Dateien im Projektverzeichnis.
`git revert 12345`	Macht den Commit mit der ID 12345 rückgängig. Git erstellt dazu einen neuen Commit, der die Änderungen des Commits 12345 aufhebt.
Branching	
Ein Branch ist ein paralleler Entwicklungszweig, in dem Sie beispielsweise experimentelle Funktionen implementieren. Die im Branch vorgenommenen Änderungen führt ein Merge wieder mit der Hauptentwicklung zusammen. Diese erfolgt im Main Branch, der standardmäßig master heißt.	
`git branch`	Existierende Branches anzeigen (den aktuellen kennzeichnet ein *).
`git branch foo`	Erstellt einen Branch mit dem Namen `foo`.
`git checkout foo`	Wechselt zum Branch `foo` (HEAD zeigt dann auf den letzten Commit im Branch `foo`).
`git checkout -b foo`	Branch `foo` anlegen und hineinwechseln.
`git branch -m "bar"`	Benennt den aktuellen Branch in `bar` um.
`git merge foo`	Fusioniert den Branch `foo` mit dem aktuellen Branch (Merging).
`git branch -d foo`	Löscht den Branch `foo`.
`git branch -D foo`	Löscht den Branch `foo`, selbst wenn die enthaltenen Änderungen noch nicht mit einem anderen Branch zusammengeführt wurden.
`git diff foo bar`	Vergleicht die Branches `foo` und `bar`.
`git diff foo bar main.c`	Vergleicht die Datei `main.c` im Branch `foo` mit ihrem Pendant im Branch `bar`.
`git log --graph --all`	Liefert ein Diagramm mit allen Branches.
`git rebase master`	Nimmt ein Rebase auf den Branch `master` vor. Der aktuelle Branch basiert dann auf dem HEAD des Branches `master`).
`git rebase --continue`	Setzt Rebase nach einem Abbruch oder Fehler fort.
Stashing	
Im Stash lassen sich Änderungen vorübergehend parken („temporäres Commit"). Damit können Sie die Arbeit an anderen Stellen fortsetzen und später zu den im Stash gemerkten Änderungen zurückkehren.	
`git stash`	Parkt alle Änderungen im Stash und dreht das Projektverzeichnis auf den letzten Commit zurück.
`git stash -u`	Parkt zusätzlich die von Git noch nicht erfassten Dateien im Stash.
`git stash --all`	Parkt alle Dateien im Stash, ebenso die in der Datei `.gitignore` ausgeschlossenen.
`git stash show`	Zeigt den Inhalt des Stashs.
`git stash apply`	Holt im Stash gemerkte Änderungen zurück.
`git stash pop`	Holt im Stash gemerkte Änderungen zurück und leert den Stash.
`git stash drop`	Leert den Stash.
Remote Repositories	
Remote Repositorys verknüpfen Sie über einen Aliasnamen mit dem lokalen Repository. In allen Git-Befehlen lässt sich dann beispielsweise `extern` anstelle von `https://example.com/user/repo.git` schreiben. Nach dem Klonen eines Repositorys kennt Git das ursprüngliche Repository unter dem Aliasnamen **origin**.	
`git remote add extern https://example.com/user/repo.git`	Verknüpft das Remote Repository `https://example.com/user/repo.git` unter dem Namen `extern` mit dem lokalen Repository.
`git remote -v`	Listet alle verknüpften Remote Repositories.
`git fetch extern`	Holt alle Branches aus dem Remote Repository `extern`.
`git fetch extern foo`	Holt nur den Branch `foo` aus dem Remote Repository `extern`.
`git merge extern/foo`	Führt den von `extern` heruntergeladenen Branch `foo` mit dem aktuellen Branch zusammen (Merge).
`git pull extern`	Bringt den aktuellen Branch auf den Stand seines Pendants im Remote Repository `extern`.
`git push extern`	Lädt den lokalen Stand in das Remote Repository `extern`.
`git push extern master`	Lädt den lokalen Stand in den Branch `master` des Remote Repositorys `extern`.