Fortgeschrittene Git-Befehle

Git ist wie ein Schweizer Taschenmesser für Entwickler –vielseitig, leistungsstark und gelegentlich verwirrend, wenn man nicht weiß, welches Werkzeug man herausziehen soll. Heute tauchen wir in einige der nützlichsten Funktionen und Workflows von Git ein: das Auswählen von Änderungen, das Zusammenführen mit Stil, das Rebasen für eine sauberere Historie, das Löschen dieser lästigen großen Dateien, die man versehentlich committe, und das Rückgängigmachen eines Commits, wenn man merkt, dass man vom richtigen Weg abgekommen ist. Lassen Sie uns das Ganze auseinandernehmen.

Cherry-Picking: Nur das Beste herausgreifen Stellen Sie sich vor, Sie haben einen Feature-Zweig mit einem Dutzend Commits, aber es gibt einen glänzenden Commit, den Sie herausgreifen und auf Ihren Hauptzweig anwenden möchten –ohne den Rest mitzunehmen. Hier kommt `git cherry-pick` ins Spiel.

Es ist super einfach: Finden Sie den Commit-Hash (Sie können ihn aus `git log` herausgreifen), wechseln Sie zu dem Zweig, auf den Sie ihn anwenden möchten, und führen Sie aus:

```
git cherry-pick <commit-hash>
```

Schon ist dieser Commit Teil Ihres aktuellen Zweigs. Wenn es zu einem Konflikt kommt, wird Git pausieren und Sie die Konflikte auflösen lassen, genau wie bei einem Merge. Sobald Sie zufrieden sind, commiten Sie die Änderungen und Sie sind fertig.

Ich benutze das ständig, wenn ein Bugfix in einen chaotischen Feature-Zweig gerät und ich ihn sofort auf `main` brauche. Seien Sie nur vorsichtig –Cherry-Picking dupliziert den Commit, sodass er einen neuen Hash erhält. Erwarten Sie nicht, dass er sich gut verhält, wenn Sie den ursprünglichen Zweig später ohne einige Aufräumarbeiten zusammenführen.

Merge-Optionen: Mehr als nur „Merge“ Merging ist Git’s Brot und Butter, aber wussten Sie, dass es verschiedene Varianten gibt? Der Standard `git merge` führt einen „Fast-Forward“durch, wenn möglich (die Historie wird gerade), oder erstellt einen Merge-Commit, wenn sich die Zweige getrennt haben. Aber Sie haben Optionen:

- `--no-ff` **(Kein Fast-Forward)**: Erzwingt einen Merge-Commit, auch wenn ein Fast-Forward möglich wäre. Ich liebe das, um eine klare Aufzeichnung zu behalten, wann ein Feature-Zweig auf `main` gelandet ist. Führen Sie es wie folgt aus:

  ```
  git merge --no-ff feature-branch
  ```

- `--squash`: Zieht alle Änderungen aus dem Zweig in einen Commit auf Ihrem aktuellen Zweig. Kein Merge-Commit, nur ein einzelnes, ordentliches Paket. Perfekt, um einen chaotischen Zweig in etwas Präsentables zu verwandeln:

```
git merge --squash feature-branch
```

Danach müssen Sie manuell commiten, um den Deal abzuschließen.

Jede hat ihren Platz. Ich neige dazu, `--no-ff` für langlebige Zweige und `--squash` zu verwenden, wenn ich einen Zweig voller „WIP“-Commits habe, die ich lieber vergessen würde.

Rebasing: Geschichte umschreiben wie ein Profi Wenn Merges zu unordentlich erscheinen, könnte `git rebase` Ihr Ding sein. Es nimmt Ihre Commits und spielt sie auf einen anderen Zweig, sodass Sie eine lineare Historie erhalten, die so aussieht, als hätten Sie alles von Anfang an perfekt geplant.

Wechseln Sie zu Ihrem Feature-Zweig und führen Sie aus:

```
git rebase main
```

Git wird Ihre Commits abheben, den Zweig aktualisieren, um `main` zu entsprechen, und Ihre Änderungen wieder oben draufsetzen. Wenn Konflikte auftreten, lösen Sie sie und führen Sie `git rebase --continue` aus, bis alles erledigt ist.

Der Vorteil? Eine makellose Zeitachse. Der Nachteil? Wenn Sie diesen Zweig bereits gepusht haben und andere daran arbeiten, schreibt das Rebasen die Historie um –willkommen bei den wütenden E-Mails von den Teamkollegen. Ich halte mich an das Rebasen für lokale Zweige oder Solo-Projekte. Für geteilte Sachen ist das Merging sicherer.

Löschen großer Dateien aus der Historie: Oops, das 2GB-Video Wir waren alle schon mal da: Sie haben versehentlich eine riesige Datei committe, gepusht und jetzt ist Ihr Repo aufgebläht. Git vergisst nicht so leicht, aber Sie können diese Datei mit etwas Aufwand aus der Historie entfernen.

Das Werkzeug der Wahl ist hier `git filter-branch` oder das neuere `git filter-repo` (ich empfehle Letzteres –es ist schneller und weniger fehleranfällig). Angenommen, Sie haben `bigfile.zip` committe und möchten es loswerden: 1. Installieren Sie `git-filter-repo` (überprüfen Sie die Dokumentation für die Einrichtung). 2. Führen Sie aus: `git filter-repo --path bigfile.zip --invert-paths` Dies entfernt `bigfile.zip` aus jedem Commit in der Historie. 3. Pushen Sie die umgeschriebene Historie mit Gewalt: `git push --force`

Achtung: Dies schreibt die Historie um, also koordinieren Sie dies mit Ihrem Team. Und wenn es in einem Pull-Request irgendwo ist, müssen Sie möglicherweise auch die Referenzen bereinigen. Sobald es weg ist, wird Ihr Repo nach einer Müllsammlung (`git gc`) schlanker.

Uncommitting: Die Uhr zurückdrehen Haben Sie einen Commit gemacht und es sofort bereut? Git hat Sie im Griff. Es gibt ein paar Möglichkeiten, dies rückgängig zu machen, je nachdem, wie weit Sie gegangen sind:

- **Wenn Sie noch nicht gepusht haben**: Verwenden Sie `git reset`. Um den letzten Commit rückgängig zu machen, aber die Änderungen in Ihrem Arbeitsverzeichnis zu behalten:

```
git reset HEAD^ --soft
```

Möchten Sie die Änderungen vollständig loswerden?

```
git reset HEAD^ --hard
```

- **Wenn Sie bereits gepusht haben**: Sie müssen die Historie umschreiben. Setzen Sie lokal mit `git reset HEAD^` zurück und pushen Sie mit Gewalt:

  ```
  git push --force
  ```

  Auch dies verändert die geteilte Historie, also gehen Sie vorsichtig vor.

Ich habe mich mit `git reset --soft` mehr als einmal gerettet –perfekt, wenn man zu früh committe und etwas anpassen muss.

Zusammenfassung Die Flexibilität von Git macht es so mächtig, aber es ist leicht, sich darin zu verheddern, wenn man seine Optionen nicht kennt. Cherry-Pick für chirurgische Präzision, passen Sie Merges an Ihren Workflow an, rebase für eine polierte Historie und geraten Sie nicht in Panik, wenn Sie einen Fehler auslöschen müssen –ob es sich um eine große Datei oder einen hastigen Commit handelt. Üben Sie dies auf einem Test-Repo, wenn Sie nervös sind, und bald werden sie Ihnen wie selbstverständlich vorkommen.

Was ist Ihr Lieblingstrick mit Git? Lassen Sie es mich wissen –ich bin immer offen für neue Erkenntnisse!