Git Magic

Beim Arbeiten mit Git Repositories sind folgende Guidelines immer sehr wichtig:

1. Niemals Secrets (Passwörter, Private Keys, SSH Keys, AWS Secret Keys, ...) unverschlüsselt im Repository speichern (besonders in public repos!!!)
   - Sicherheitsrisiko!
2. Niemals 3rd Party Dependencies von Paket managern (zB. node_modules, vendor / php composer, ruby gems, etc.) in das Repository commiten!
   - Dafür gibt es meistens eine Konfigurationsdatei (package.json und package-lock.json), wo genau drin steht, welche Dependencies gebraucht werden
   - Diese Dependency Graphs können sehr schnell anwachen und auch für ein kleines Projekt hunderte von MBs darstellen
   - Bei jedem git clone müssen nun sehr viele Daten gedownloaded werden
   - Es gibt häufig Speicherplatz Limitierungen bei Github / Gitlab
   - **Ist in der Regel ein Zeichen für einen Anfänger, also vorallem bei eventuellen Arbeitgebern vermeiden**

Wie verhindert man solche Fälle?

Um solche Fehler grundlegend zu vermeiden, benutzt man ein **.gitignore** File!

Diese Datei gilt immer für den aktuellen Ordner und es ist Best Practice im root des Repositories so eine Datei anzulegen. Alle Dateinamen die hier angegeben werden, werden nicht mit einem `git add` berücksichtigt. Das funktioniert aber nur, **wenn die Dateien NOCH NICHT zur Git History hinzugefügt wurden**

Hier ist die Dokumentation zu finden: https://git-scm.com/docs/gitignore

Folgend ein Beispiel wie man das *node_modules* Verzeichnis auschließen kann:

```
node_modules/
```

Wie schafft man es diese Fehler Rückgangig zu machen?

In den folgenden Beispielen, gehen wir nun also davon aus, dass ausversehen der *node_modules* Ordner in unser git Repo gelangt ist:

Nach hinzufügen zum Staging Bereich ABER NOCH NICHT COMMITED

Dieser Fall ist relativ einfach und wir auch mit dem `git status` Befehl angezeigt:

```
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   node_modules/.bin/_mocha
        new file:   node_modules/.bin/flat
        new file:   node_modules/.bin/he
        new file:   node_modules/.bin/js-yaml
        new file:   node_modules/.bin/mime
        new file:   node_modules/.bin/mocha
```

Das heißt man kann die node_modules wie folgt wieder aus der Staging Area entfernen:

```
git restore --staged node_modules
```

Nach Commit aber noch nicht gepusht

Da der Fehler sonst immer noch in der Git History vorhanden ist (ergo `git clone` immer noch sehr lange geht) ist es wichtig den eigentlichen Commit wo der Fehler passiert ist abzuändern. Ein nachträgliches abändern und danach neu commiten macht daher keinen Sinn.

Fehler war im letzten Commit

Hier gibt es mehrere Möglichkeiten:

1. Fehler ausgleichen und dann den Commit nachträglich abändern:

```
rm -rf node_modules # Löscht den node_modules Ordner
git add . # Staged die Löschung der node_modules
git commit --amend # Ändert den letzten Commit ab, es öffnet sich ein Editor und man kann die Commit msg abän
```

2. Den letzten Commit verwerfen und die Änderungen wieder zum Staging hinzufügen:

```
git reset --soft HEAD^ # Verwirft den letzten commit (--soft behält die Änderungen im Staging)
git restore --staged node_modules # Wie im ersten Beispiel // entfernt den Ordner aus dem Staging bereich
git commit -m 'Eure alte commit msg' # Commited die richtigen Changes (ohne die Node Modules)
```

Fehler war in Commit vor letztem Commit

Die nächste Möglichkeit ist eher fortgeschritten aber kann einem das Leben in vielen Situationen erleichtern.

Das Stichwort hier ist Interactive Rebasing: Bitte schaue dir für mehr Informationen die offizielle Dokumentation an:

https://git-scm.com/book/en/v2/Git-Tools-Rewriting-History#_changing_multiple

Zuerst müssen wir herausfinden, in welchem commit der Fehler passiert ist. Das geht zum Beispiel mit einem Commandline tool wie `tig`.

Wenn wir also sehen der Fehler war in dem vorletzten Commit:

```
git rebase -i HEAD~2 # -i für interactive und HEAD~2 für den 2 Commits zurück
```

Dann öffnet sich ein Editor mit folgendem Inhalt:

```
pick 9e18621 feat: add rechner.js and node_modules by mistake
pick a142697 test: add unit tests for rechner

# Rebase d9849d1..a142697 onto d9849d1 (2 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup [-C | -c] <commit> = like "squash" but keep only the previous
#                    commit's log message, unless -C is used, in which case
#                    keep only this commit's message; -c is same as -C but
#                    opens the editor
# x, exec <command> = run command (the rest of the line) using shell
# b, break = stop here (continue rebase later with 'git rebase --continue')
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
# m, merge [-C <commit> | -c <commit>] <label> [# <oneline>]
# .       create a merge commit using the original merge commit's
# .       message (or the oneline, if no original merge commit was
```

```
# .       specified); use -c <commit> to reword the commit message
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
```

Hier sehen wir jetzt unsere letzen 2 commits und können diese Bearbeiten und dann Speichern und Schließen:

```
edit 9e18621 feat: add rechner.js and node_modules by mistake
pick a142697 test: add unit tests for rechner

# Rebase d9849d1..a142697 onto d9849d1 (2 commands)
#
```

Dann wird jetzt folgende Hilfestellung angezeigt:

```
Stopped at 9e18621...  feat: add rechner.js and node_modules by mistake
You can amend the commit now, with

  git commit --amend

Once you are satisfied with your changes, run

  git rebase --continue
```

Das heißt jetzt sind wir bei genau der Stelle in unserer Commit History und können diese wieder mit `git commit --amend` bearbeiten:

```
git rm --cached -r node_modules/ # oder einfach rm -rf node_modules
git commit --amend
git rebase --continue
# Es kann sein, das es jetzt Merge Conflikte gibt, die behoben werden müssen
```

Wenn die Changes bereits gepusht wurden

Die nachfolgenden Änderungen funktionieren nur, wenn euer Branch in Github nicht protected ist, falls doch geht es leider nicht Git push --force und --force-with-lease gefährlich wenn mehrere Leute auf einem Branch zusammen arbeiten!

Im Grunde könnt ihr den Commit wie im letzten Schritt gezeigt überarbeiten und dann einfach mit folgendem Befehl pushen. **Ein normaler push reicht nicht aus, da ihr die Git History umgeschrieben habt**

```
git push --force-with-lease
```

Aufgabe zum Testen

Um das gelernte zu testen, kannst du (freiwillig) folgende Schritte durchführen:

1. Erstelle ein neues Repo und initialisiere ein nodejs projekt:

```
mkdir project
cd project
git init
npm init -y
git add .
git commit -m 'init commit'
```

2. Installiere ein Paket, erstelle eine neue Js Datei und commite alles (ausversehen auch node_modules):

```
npm i express
echo "console.log('Hallo');" > index.js
git add .
git commit -m 'feat: add js file'
```

3. Editiere die index.js Datei und füge ein Datenbank Secret ein, commite auch diese Änderungen:

```
//index.js
console.log('Hallo');
const DB_STRING = 'Provider=PostgreSQL OLE DB Provider;Data Source=myServerAddress;location=myDataBase;User I
```

```
git add .
git commit -m 'feat: added db stuff'
```

4. Mache diese Fehler rückgängig, sodass sie auch nicht mehr in der Git History vorhanden sind!