Git-Workflow

Allgemein

Diese Seite gibt einen Überblick darüber, wie wir in der SQUEEZE-Entwicklung mit Git arbeiten: welche Branches es gibt, wofür sie da sind und wie Änderungen ihren Weg in ein Release finden.

Der Umfang unserer Git-Projekte ist unterschiedlich, deshalb gelten nicht überall die gleichen Anforderungen – je kleiner das Projekt, desto einfacher und weniger strikt der Workflow. Der hier beschriebene Workflow gilt für **SQUEEZE und das DEXP-Frontend**.

Das Repository liegt in Azure DevOps im Projekt *DEXPRO Platform*:

```
git clone git@ssh.dev.azure.com:v3/DEXPRO/DEXPRO%20Platform/SQUEEZE
```

Branch-Modell

Wir arbeiten mit einem langlebigen Entwicklungs-Branch und pro Minor-Version einem Release-Branch:

Branch	Bedeutung
`develop`	Aktueller Entwicklungsstand und Ausgangspunkt für neue Arbeit. Der Branch *soll* möglichst immer lauffähig sein.
`release/<major.minor>` (z. B. `release/2.34`)	Enthält die getaggten Releases einer Minor-Version. Jeder getaggte Commit *muss* lauffähig sein. Hier entstehen Patch-Releases und Hotfixes. Release-Branches werden **nicht** pauschal in den `develop` zurückgemerged.
Themen-Branches (`feature/`, `task/`, `fix/`, `bugfix/`, `hotfix/`)	Kurzlebige Branches für eine konkrete Änderung. Sie werden per Pull Request in `develop` oder einen `release/`-Branch gemerged und danach gelöscht.

Namenskonvention für Branches

Branch-Namen werden auf Englisch formuliert und beginnen mit der Kategorie, gefolgt von der Work-Item-/Ticketnummer und einer kurzen Beschreibung:

```
feature/18157-image-tools-refactor
task/17609-add-savon-cli-tool
fix/17470-log-fix
```

Ist ein Branch gezielt für eine bestimmte Version gedacht, wird die Version mit aufgenommen – so ist auf einen Blick erkennbar, wohin die Änderung gehört:

```
hotfix/2.33-18092-createXMLErrorPDF-generates-unique-pdf
fix/2.32/kosit-pdf-creation-crash
```

Commits

- Commit-Messages werden auf Englisch formuliert.
- Commits möglichst so schneiden, dass eine zusammenhängende Änderung in einem Commit liegt. Das ist nicht nur für das Review angenehmer, sondern vor allem die Grundlage dafür, dass ein Fix später sauber in einen anderen Branch ge-cherrypickt werden kann.

Änderungen einbringen

Änderungen kommen grundsätzlich über einen **Pull Request** in `develop` oder einen `release/`-Branch – kein direkter Push auf diese Branches. Der PR ist der Ort für Code-Review, Build- und Testlauf.

Der typische Ablauf für ein neues Feature oder eine Task:

1. Aktuellen `develop` auschecken und aktualisieren.
2. Themen-Branch nach obiger Konvention anlegen.
3. Entwickeln, committen, Branch pushen.
4. Pull Request nach `develop` erstellen und mit dem Work Item verknüpfen.
5. Nach Review und grünem Build mergen; der Branch wird anschließend gelöscht.

Bugfixing über mehrere Versionen

Beim Bugfixing kommt neben der Problemanalyse die Frage dazu, **auf welchem Produktstand** der Bug auftritt und in welchen Versionen er behoben werden muss. Im Idealfall wird der verursachende Commit identifiziert – daraus ergibt sich, welche Fixes wo angebracht sind.

Muss ein Fix rückwirkend in eine ausgelieferte Version, sollte wie folgt vorgegangen werden:

- **Start auf dem `release/`-Branch:**
  - Fix vom betroffenen `release/`-Branch abzweigen
  - Problem beheben und commiten
  - Per PR in den Release-Branch zurückbringen
- **Backmerge in nachfolgende Release-Branches/Develop:**

    - Danach muss der fix in alle folgenden Release-Branches bzw den Develop gemerged werden (z.B. `release/2.33` -> `release/2.34` -> `develop`)
    - Dazu wird zunächst der Ziel-Branch, auf den gemerged werden soll ausgewählt (nächster Release-Branch oder develop)
    - Davon wird ein neuer Branch erstellt (`task/XXXX-backmerge-release-xy`)
    - Der Release-Branch wird auf den neu erstellten Branch gemerged und alle Konflikte behoben.
    - Der Backmerge-Branch wird gepusht und ein PR erstellt

Auf diese Weise ist sichergestell, dass ein Bugfix, der im Release landet, auch bis auf den aktuellen develop durchgereicht wird und nicht unterwegs "verloren geht".

Weiterführend

- Code Reviewing und Pull Requests
- Versionierung
- Release und Deployment

Squeeze-Release Arbeiten

Diese Seite beschreibt die **Git-Seite eines neuen SQUEEZE-Releases**: vom Abzweigen des Release-Branches bis zu dem Punkt, an dem der Release-Candidate auf Staging deployed werden kann und der `develop` wieder auf die Folgeversion zeigt.

Die Schritte sind bewusst als Checkliste formuliert – am besten von oben nach unten abarbeiten und nichts überspringen, da spätere Schritte auf früheren aufbauen.

Beteiligt sind zwei Repositories:

Repository	Inhalt	Branch	Tag
SQUEEZE	Backend	`release/2.XX`	`2.XX.0-rc.N`
DEXPRO Platform	Frontend	`release/squeeze/2.XX`	`squeeze-2.XX.0-rc.N`

In den Beispielen unten wird die Version **2.34** released, der `develop` läuft danach auf **2.35** weiter.

1. Release-Branch abzweigen

☐ **Abzweig-Commit auf `develop` bestimmen.** Der Commit muss so gewählt sein, dass genau die Features enthalten sind, die im Azure Sprint-Board zur Release-Nummer gehören – und **keine** Features, die erst für die nächste Version vorgesehen sind.

- Schaue auf Azure Commits Seite
- Schaue nach den Merge-Daten ungefähr auf Start des Sprints
- Schaue dir das Sprint-Board des neuen Sprints an
- Guckst du was da drauf ist und nicht "super-extra-wichtig-will-ich-dringend-noch-haben-sonst-geht-Welt-unter"
- Ermittle Schnittmenge daraus und finde den letzten Commit der nichts aus dem neuen Sprint behandelt

☐ Im **SQUEEZE**-Repo den Branch `release/2.34` von diesem Commit anlegen.

☐ Im **DEXPRO Platform**-Repo den passenden Branch `release/squeeze/2.34` anlegen.

> Der Abzweigpunkt ist die fehleranfälligste Stelle im gesamten Prozess. Ein zu spät abgezweigter Branch nimmt Features der Folgeversion mit ins Release. Im Zweifel vor dem Anlegen des Branches mit dem PO bzw. dem Team abgleichen, welche Work Items zum Release gehören.

2. Vorherigen Release-Branch mergen

Als Absicherung, falls ein Backmerge des letzten Releases in den `develop` vergessen wurde:

- ☐ Im SQUEEZE-Repo `release/2.33` in `release/2.34` mergen.
- ☐ Im DEXPRO Platform-Repo `release/squeeze/2.33` in `release/squeeze/2.34` mergen.

Bringt der Merge keine Änderungen, ist alles in Ordnung – der Schritt kostet nichts und verhindert, dass ein Hotfix aus dem letzten Release im neuen Release wieder fehlt.

3. Release-Candidate taggen

Getaggt wird nach Semver, beginnend mit `rc.1`; jeder weitere Kandidat zählt hoch (`rc.2`, `rc.3`, …).

- ☐ SQUEEZE-Repo (Backend): Tag `2.34.0-rc.1` auf dem Release-Branch setzen.
- ☐ DEXPRO Platform-Repo (Frontend): Tag `squeeze-2.34.0-rc.1` auf dem Release-Branch setzen.

> Das Frontend-Tag trägt zwingend das Präfix `squeeze-`, weil im DEXPRO Platform-Repo mehrere Produkte liegen.

4. Images bauen

Die Pipelines starten für Tags nicht automatisch, sie müssen in Azure DevOps **manuell auf dem jeweiligen Tag** gestartet werden:

- ☐ build and test - squeeze & worker (e2e) für das Tag im SQUEEZE-Repo → erzeugt die Docker-Images für Squeeze und Worker.
- ☐ push image - dexp-frontend für das Tag im DEXPRO Platform-Repo → erzeugt das Frontend-Image.
- ☐ Beide Pipelines sind grün durchgelaufen.

Damit ist der Release-Candidate fertig und kann auf Staging deployed werden.

5. `develop` auf die nächste Version heben

Direkt nach dem Abzweigen muss der `develop` im **SQUEEZE**-Repo auf die Folgeversion gestellt werden – sonst laufen die nächsten Entwicklungs-Builds weiter unter der alten Versionsnummer. Es gibt dafür im aktuellen Sprint eine Story für die Nacharbeiten, die in neues Sprints dupliziert wird, z.B. für 2.34 Release Nacharbeiten

- [ ] `config/versions.json` auf die neue Version setzen:

```
{
  "initialVersion": "2.35-develop"
}
```

- [ ] In `devops/swagger/ts-client.config.json` die Version erhöhen **und die Build-Nummer auf 1 zurücksetzen**:

```
{
  "npmName": "@dex/squeeze-client-ts",
  "npmVersion": "2.35.0-build.1"
}
```

- [ ] Beide Änderungen in **einem** PR nach `develop` bringen und mergen.

> Die Build-Nummer nicht weiterzählen lassen, sondern auf `build.1` zurücksetzen. Sie zählt innerhalb einer Version die veröffentlichten TS-Client-Pakete.

6. Neue TS-Client-Version verteilen

Nach dem Merge des PRs aus Schritt 5 wird automatisch eine neue Version des TypeScript-Clients `@dex/squeeze-client-ts` gebaut und veröffentlicht. Diese Version muss anschließend in den abhängigen Projekten eingetragen werden – in dieser Reihenfolge:

- [ ] **API-Tests** im SQUEEZE-Repo (`test/apitests/package.json`)
- [ ] SQUEEZE Viewer
- [ ] SQUEEZE Configframework
- [ ] **Frontend** im DEXPRO Platform-Repo

Die Reihenfolge ist wichtig: Viewer und Configframework werden vom Frontend eingebunden, das Frontend kommt deshalb zuletzt.

Weiterführend

Diese Seite deckt nur die Git- und Build-Schritte ab. Die weiteren Release-Tätigkeiten – finale Versionsnummer im Release-Branch, On-Premise-Artefakte, Testszenarien für den RC sowie Deployment und Ansible-Rollout – sind im DEXPRO-Platform-Wiki dokumentiert:

- Squeeze Releases
- Release und Deployment
- Versionierung

Zum grundsätzlichen Branch-Modell siehe Allgemein.