



GIT DEPLOYMENTS
DONE RIGHT?

by David Danier

1

Kurzprofil

2

Wieso git beim Deployment?

3

Was fehlt?

4

Allgemein: git für das Deployment

5

Unser Vorgehen – git Deployment done right

6

Fazit

1

Kurzprofil

2

Wieso git beim Deployment?

3

Was fehlt?

4

Allgemein: git für das Deployment

5

Unser Vorgehen – git Deployment done right

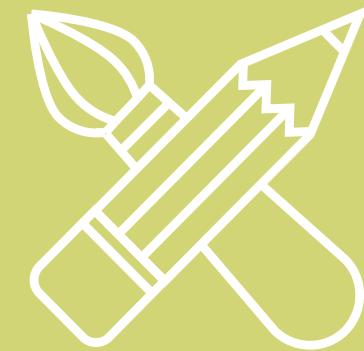
6

Fazit

**DAVID DANIER**

Supervising Chief Technical Officer of
Scalable Customer Success

AGENTURLEISTUNGEN



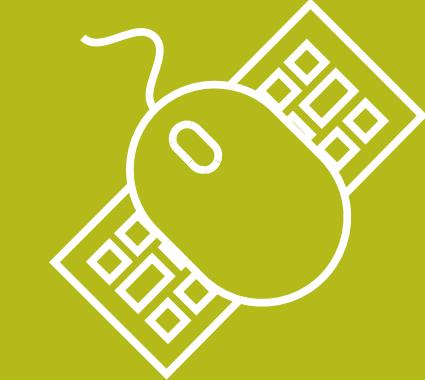
INTERFACE DESIGN

User Interface
Mobile Design
User Experience



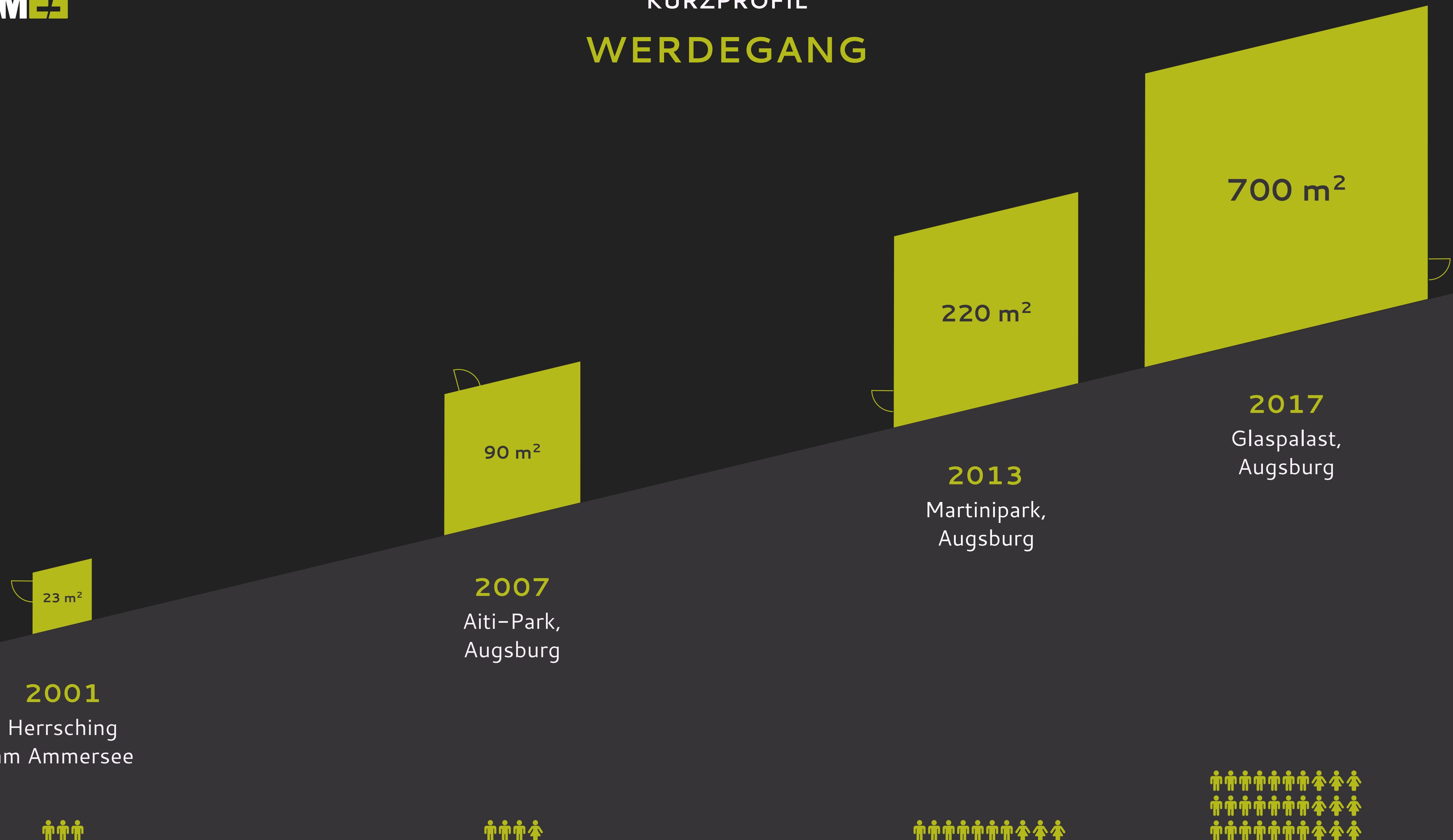
WEBDEVELOPMENT

Webanwendungen
Content Management
E-Commerce



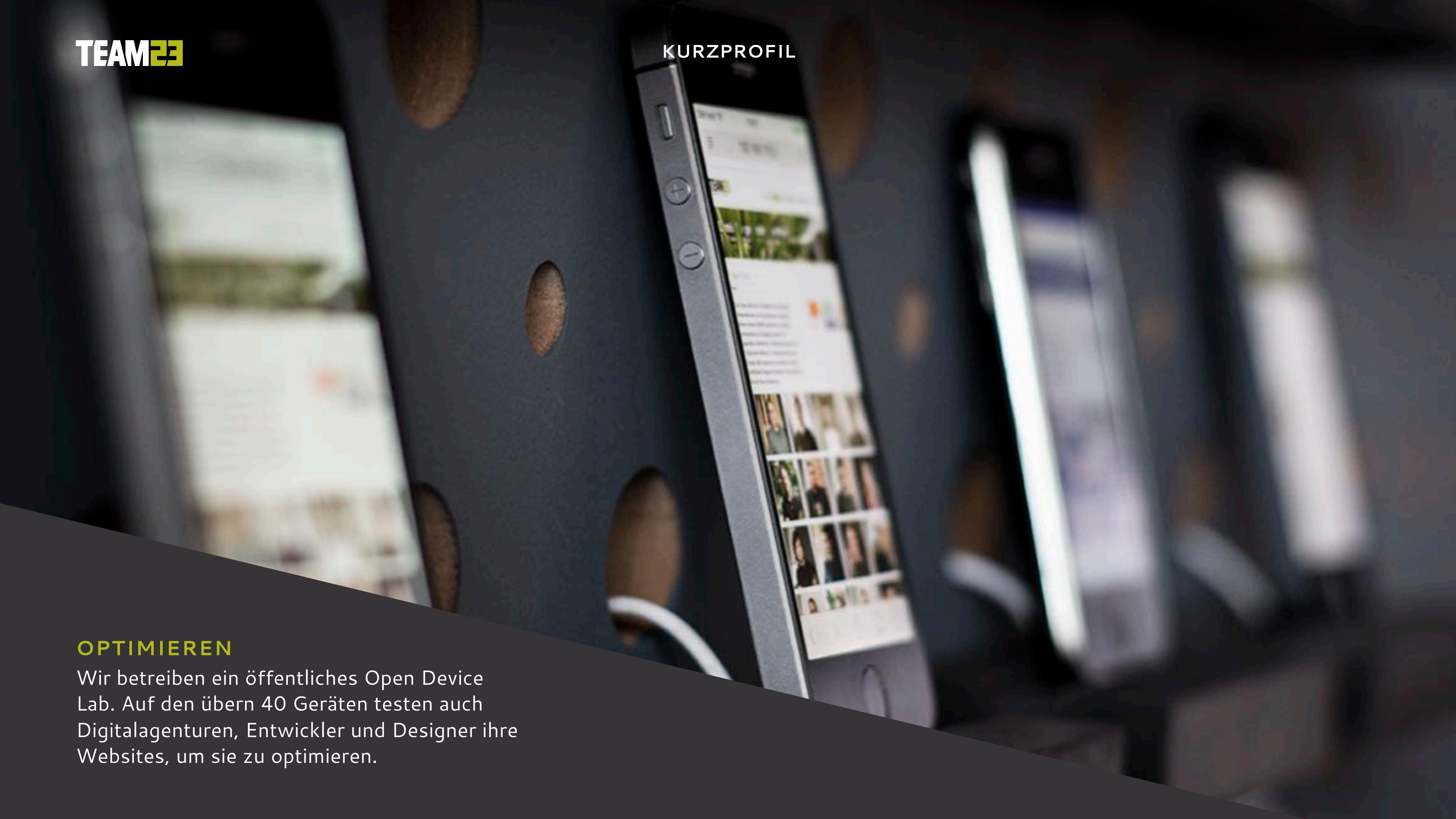
ONLINE-MARKETING

Strategie & Beratung
Suchmaschinenoptimierung (SEO)
Suchmaschinenmarketing (SEA)

WERDEGANG

OPTIMIEREN

Wir betreiben ein öffentliches Open Device Lab. Auf den übern 40 Geräten testen auch Digitalagenturen, Entwickler und Designer ihre Websites, um sie zu optimieren.



1

Kurzprofil

2

Wieso git beim Deployment?

3

Was fehlt?

4

Allgemein: git für das Deployment

5

Unser Vorgehen – git Deployment done right

6

Fazit

WIESO GIT FÜR DEPLOYMENTS INTERESSANT IST

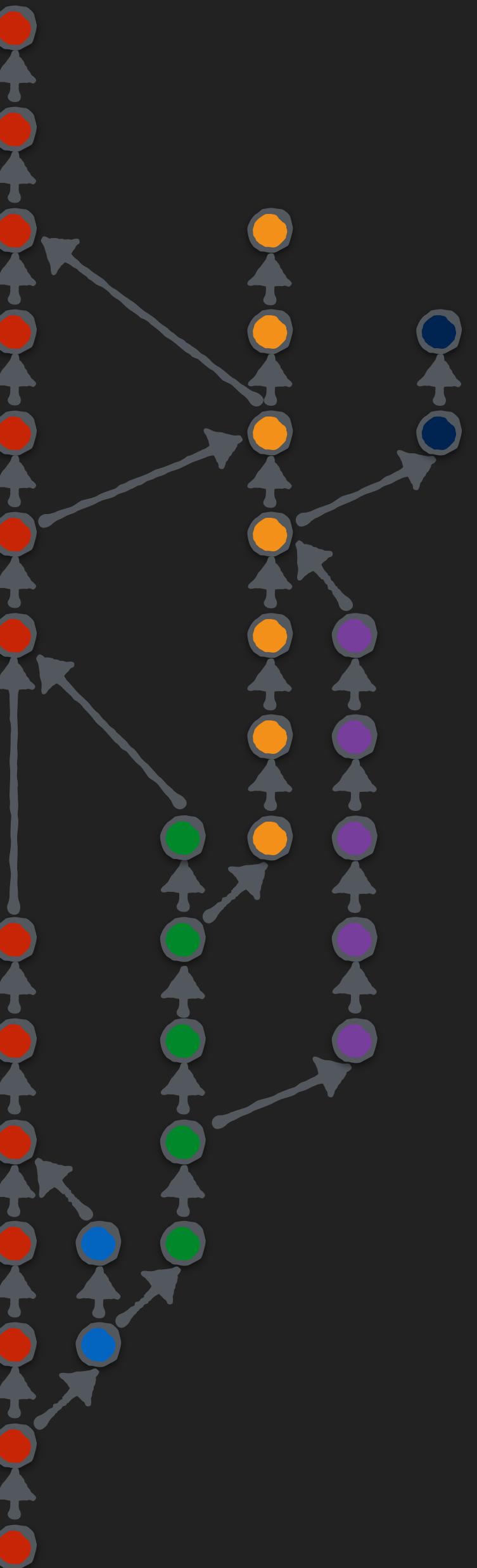
WARUM GIT?

- Keine Dateien können vergessen werden
- Änderungen sind nachvollziehbar
- Es ist möglich auf eine alte Version zurück zu wechseln
- Dezentrales System, dadurch auch Cluster-Deployment gut abbildbar
- Zusätzliche Sicherheit, Manipulationen können erkannt werden
- Folgt der Entwicklung, keine Parallelwelt
- Stellt gleichzeitig sauberen Workflow sicher (z.B. Trennung von Entwicklungssystem zu Server notwendig Passwörter im Repo klappt nicht)

PARALLELE VERSIONSZWEIGE

BRANCHES

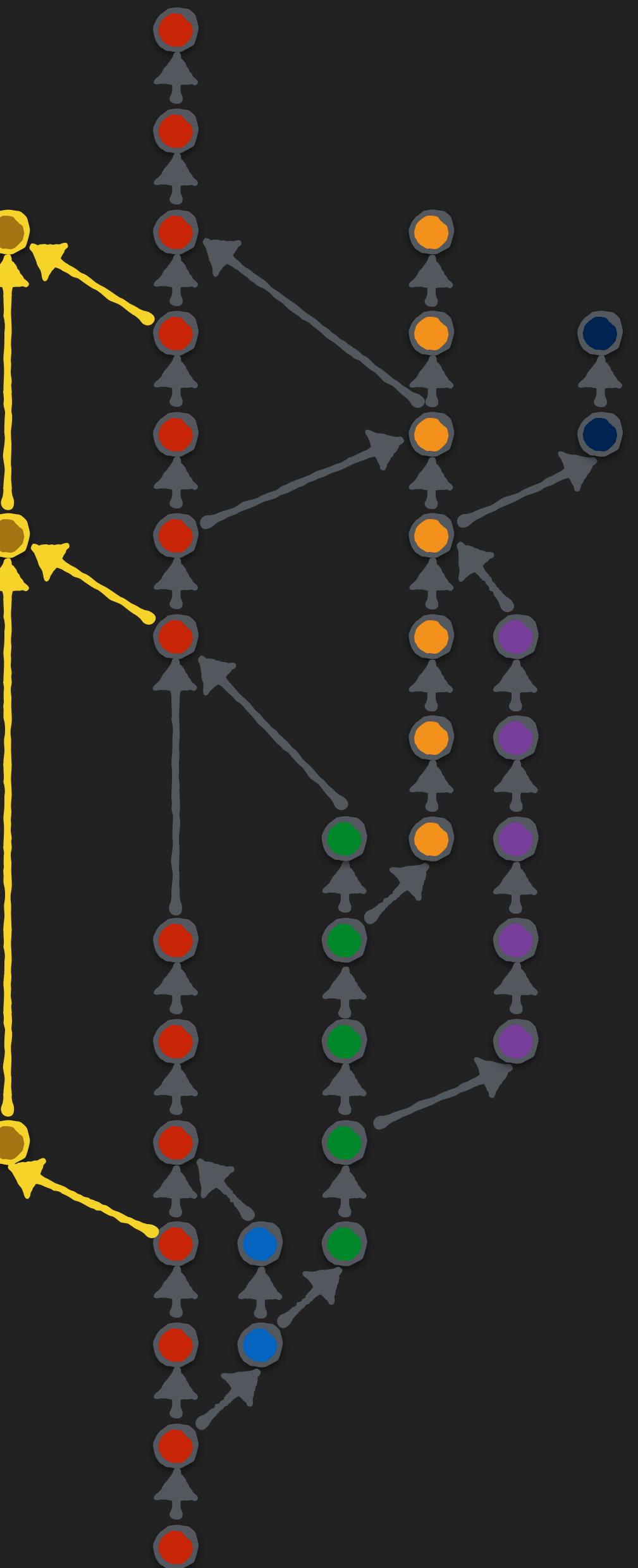
- Unterscheidung verschiedener Versionszweige wird ermöglicht
- Feature-Branches ermöglichen strukturierte Entwicklung
- Gleichzeitig verschiedene Deployment-Ziele (testing, staging, production) unterscheidbar
- Somit Grundlage für einen sinnvollen Deployment-Prozess

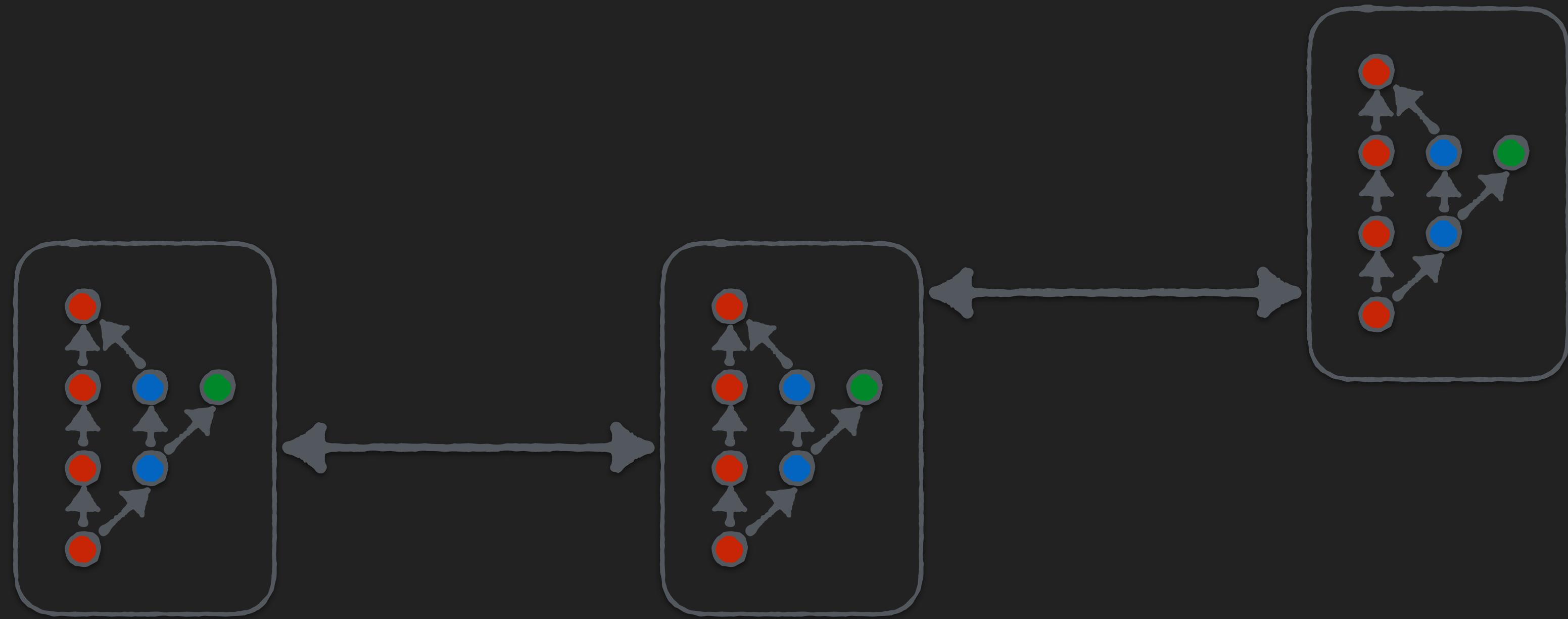


PARALLELE VERSIONSZWEIGE

BRANCHES

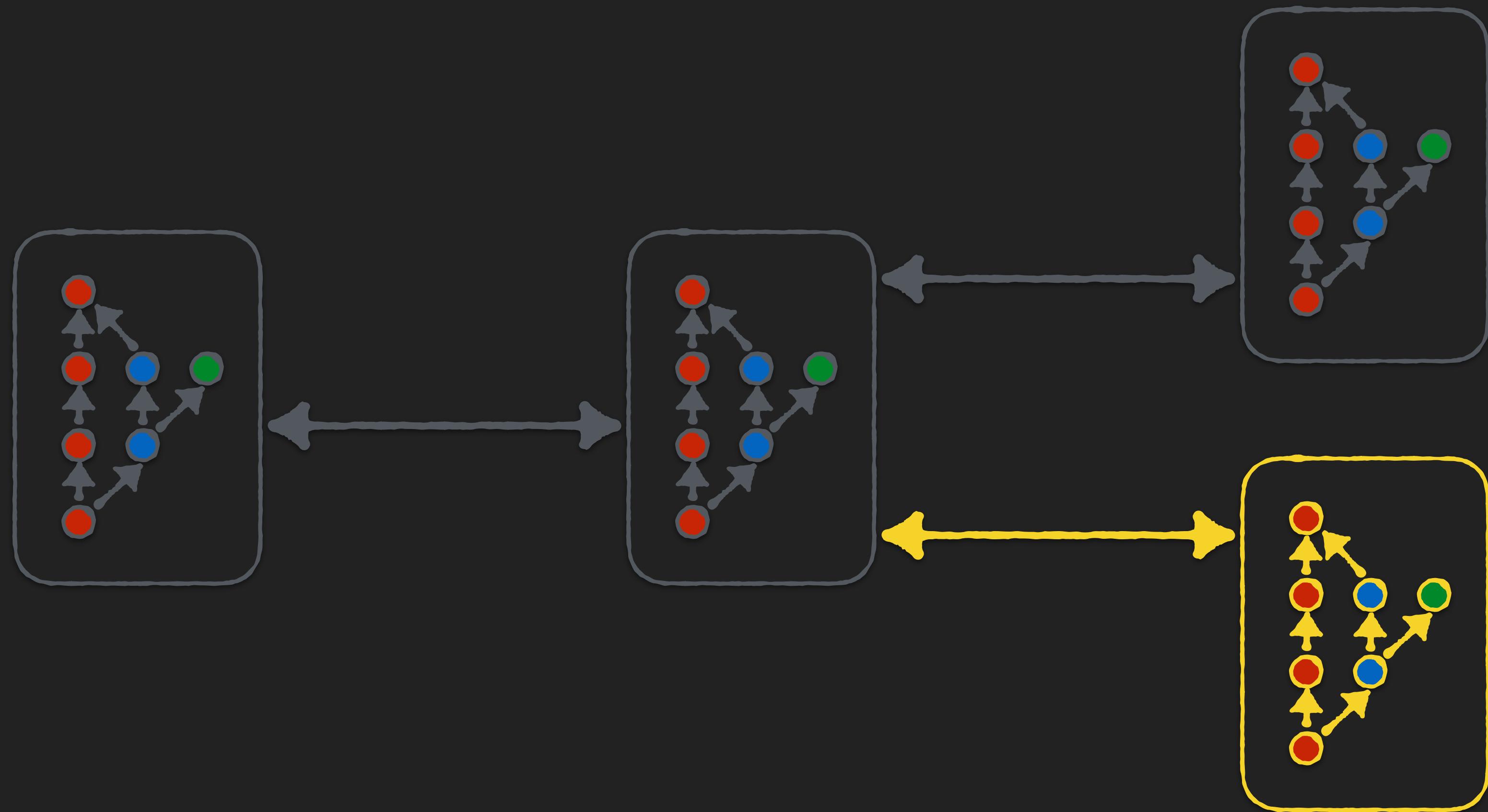
- Unterscheidung verschiedener Versionszweige wird ermöglicht
- Feature-Branches ermöglichen strukturierte Entwicklung
- Gleichzeitig verschiedene Deployment-Ziele (testing, staging, production) unterscheidbar
- Somit Grundlage für einen sinnvollen Deployment-Prozess





DEZENTRALE RESITORIES

Beim Deployment wird der Server wie ein normales Repository behandelt. (kein Sonderfall)



DEZENTRALE RESITORIES

Beim Deployment wird der Server wie ein normales Repository behandelt. (kein Sonderfall)

1

Kurzprofil

2

Wieso git beim Deployment?

3

Was fehlt?

4

Allgemein: git für das Deployment

5

Unser Vorgehen – git Deployment done right

6

Fazit

WIESO GIT VIELLEICHT PROBLEME BEREITEN KANN

WAS FEHLT BEI GIT – WAS IST SCHWIERIG

- Kein Build-Prozess, alle notwendigen Dateien müssen in git liegen (oder der Server wird komplexer)
- Nicht alle Dateien sind zum Betrieb notwendig (z.B. bei AngularJS2-Projekten)
- Nur eingeschränkte Möglichkeit Dateirechte zu tracken
- Verzeichnisse normal nicht einzeln in git enthalten (.gitkeep? – Urgs...)
- SSH-Login notwendig
- Kritisch: .git/ darf **niemals** im Web erreichbar sein!

WIE EIGENTLICH SIEHT DAS OPTIMALE DEPLOYMENT AUS?

ANFORDERUNGEN

- Automatisiertes Einspielen der Änderungen
(damit: keine Möglichkeit etwas zu vergessen)
- Nachvollziehbarkeit aller Änderungen
(Optimal: mit Crypto möglich)
- Manipulationssicherheit
- Durchführung von: Migrationen, Cache-Aktualisierung/Leerung, Wartungsmodus, Build-Prozesse (z.B. SASS), ...
- Möglichst kurze Ausfallzeiten – optimaler Prozess zur Sicherstellung
- Abbruch im Fehlerfall bzw. auch Revert
- Keine bis wenig Angriffsfläche für „menschliches Versagen“
- Auch für High-End-Anforderungen abbildbar (Cluster, VPN, ...)
- Kompatibel für eine Vielzahl von Systemen/Konfigurationen (bei uns: Django, Drupal, TYPO3, Node, Magento, statische Webseiten, ...)

1

Kurzprofil

2

Wieso git beim Deployment?

3

Was fehlt?

4

Allgemein: git für das Deployment

5

Unser Vorgehen – git Deployment done right

6

Fazit

HEROKU, AWS, AZURE, ...

UNTERSTÜTZT DURCH HOSTING

Vorteile

- Funktioniert Out-Of-The-Box
- Wenig Know-How für Setup notwendig
- Gute Anbindung an CI-Server

Nachteile

- Vendor-Lock – Abhängigkeit vom Hosting-Partner
- Teilweise auf bestimmte Systeme eingeschränkt
- Bei Problemen muss man oft viel Wissen aufholen

✓ Automatic deploys from GitHub are enabled
Every push to `master` will deploy a new version of this app. Deployes happen automatically: be sure that this branch in GitHub is always in a deployable state and before you push.

Powered by Heroku

HERE YOU CAN SEE AWS, AZURE, ...

UNTERSTÜTZT DURCH HOSTING

Vorteile

- Funktioniert Out-Of-The-Box
- Wenig Know-How für Setup notwendig
- Gute Anbindung an CI-Services

Nachteile

- Vendor-Lock – Abhängigkeit vom Hosting-Anbieter
- Teilweise nur für bestimmte Systeme eingeschränkt
- Bei Problemen muss man viel Wissen aufholen

Automatic deployments from GitHub are enabled
Every push to the master will deploy a new version of this app. Deployments happen automatically: be sure that this branch in GitHub is always in a deployable state before you push.

based on your

DER SERVER IST NORMALER CLIENT

GIT PULL

Vorteile

- Entspricht am ehesten dem normalen git Workflow
- Sehr einfaches Setup
- Weiterhin direktes Arbeiten auf dem Server möglich (There will be Dragons)

Nachteile

- git-Repository muss durch den Server erreichbar sein
- Server hat im Zweifel Zugriff auf komplette Historie
- Durch seine Einfachheit leicht zu missbrauchen (Yeah, Dragons)
- War so schon mit SVN o.ä. möglich

DER SERVER IST NORMALER CLIENT

GIT FETCH + CHECKOUT

Vorteile

- Erweiterung vom normalen „git pull“
- Dadurch ähnliche Vorteile
- Besser: Netzwerk-Übertragung getrennt vom Anwenden der Änderungen

Nachteile

- Wie bisher

DER SERVER IST NORMALES REPOSITORY

GIT PUSH

Vorteile

- Der Server ist wie bei FTP rein als Ziel/Server verwendet (keine Client-Funktion)
- Kein direkter Zugriff auf das Repository notwendig
- Klare Trennung, dadurch viel geringeres Potential etwas falsch zu machen
- Gut in Travis/... integrierbar

Nachteile

- Direkter Push ins Arbeitsverzeichnis nur mit Fingerspitzengefühl ohne Probleme machbar
- Weiterhin komplette Historie auf dem Server

DER SERVER IST NORMALES REPOSITORY

GIT PUSH + CHECKOUT

Vorteile

- Erweiterung vom normalen „git push“
- Keine direkten Änderungen im Arbeitsverzeichnis
- Wie beim „git fetch“ Trennung von Upload und Anwenden der Änderungen möglich

Nachteile

- Da ist immer noch was mit der kompletten Historie

DER SERVER IST NORMALES REPOSITORY

GIT PUSH + CHECKOUT

Zu beachten

- Durch die Automatisierung sollten beim „git checkout“ keine Fehler auftreten
- Ein „git reset“ vor dem „git checkout“ kann dies sicherstellen
- Zudem muss man mit zusätzlichen oder gelöschten Dateien im Arbeitsverzeichnis aufpassen
- Dies gilt natürlich auch für den „git fetch“ + „git checkout“

STATISCHER UMGANG

GIT ARCHIVE

Vorteile

- Einführung eines Build-Prozesses möglich
- Statische Dateien, keinerlei Logik
- Keine Historie auf dem Server

Nachteile

- Vorteile von git gehen verloren

1

Kurzprofil

2

Wieso git beim Deployment?

3

Was fehlt?

4

Allgemein: git für das Deployment

5

Unser Vorgehen – git Deployment done right

6

Fazit

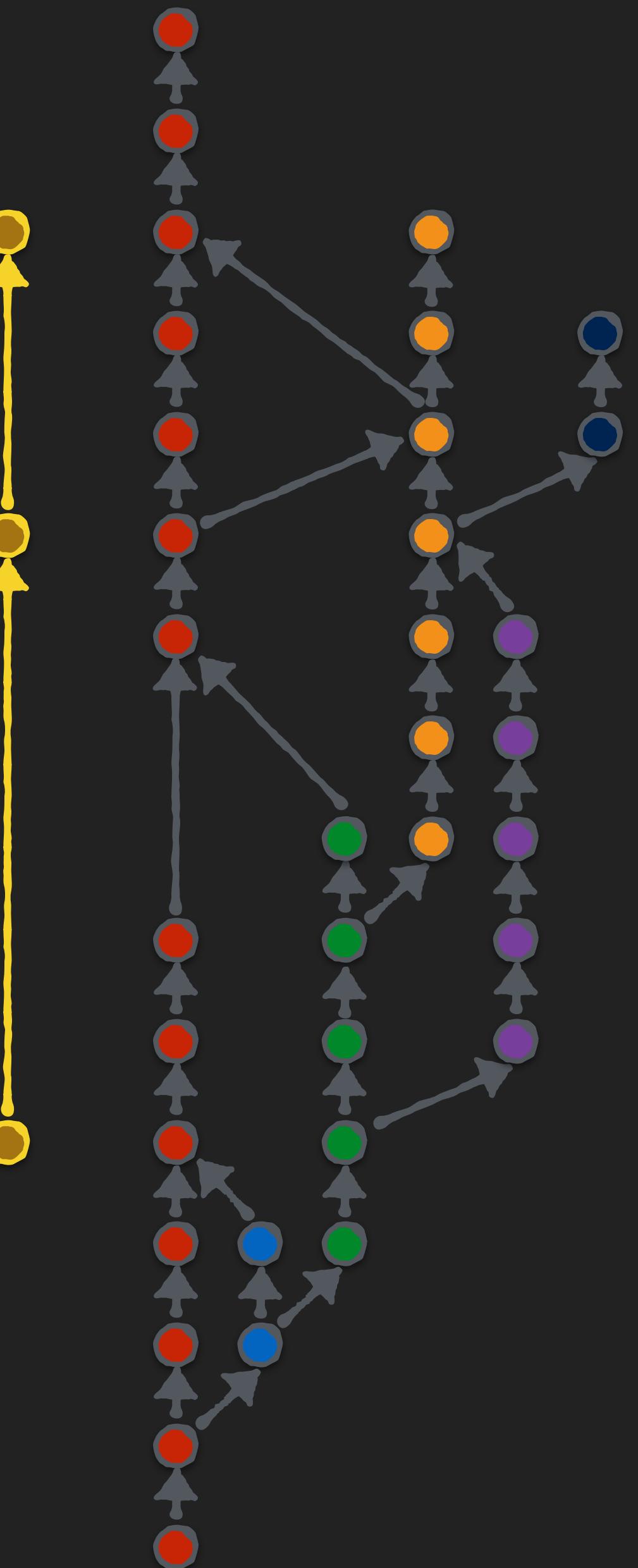
GRUNDLAGEN

DER OBJECT-STORE

- git arbeitet eher wie ein Dateisystem als klassische Versionsverwaltungssysteme
- Bedeutet: Es werden keine Unterschiede zwischen Dateien gespeichert sondern Snapshots verwendet
- Hierzu wird jede Datei innerhalb der Historie unter dem SHA1-Hashes des Dateiinhalts abgelegt
(z.B. .git/objects/4b/
3b97b69e308c4a9da47e860271235a9d147c3c)
- Ändert sich also eine Datei-Inhalt nicht, so ändert sich der Hash nicht und es wird die gleiche Dateiname im Object Store „berechnet“
- Dadurch benötigen nur geänderte Dateien zusätzlichen Speicherplatz
- Zusätzliche Komprimierung zur effizienteren Nutzung des Speicherplatzes
- Hinweis: Auch Verzeichnisinhalte (Tree) und Commits sind auf diese Weise abgelegt (Hash über „Datei“-Inhalt)

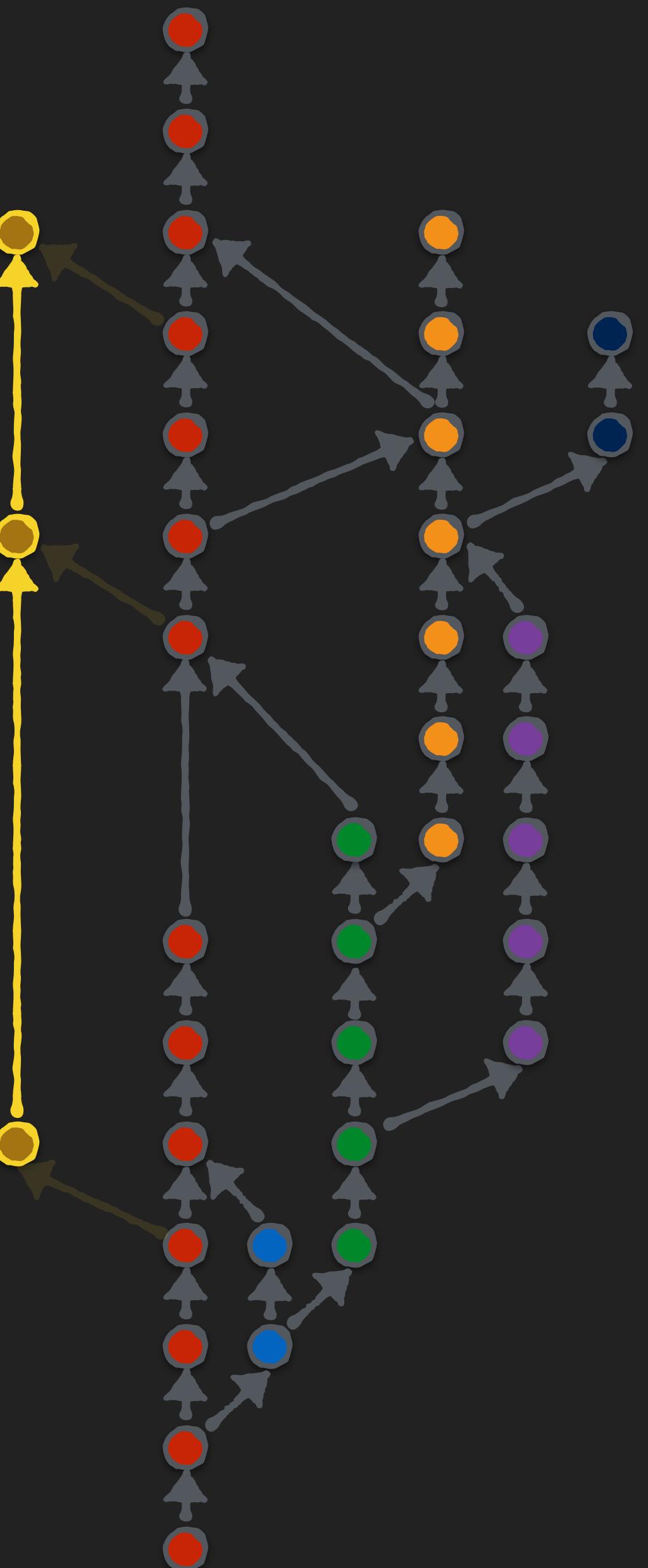
GETRENNTER VERSIONSZWEIG**RELEASE-
BRANCH**

- Komplett getrennter
Versionszweig
(production → release/production)
- Wird aus dem git-Tree-Objekt
generiert
(Low-Level-Bibliothek)
- Kein Bezug zur normalen Historie,
keine Merges zum Release-
Branch



GETRENNTER VERSIONSZWEIG**RELEASE-
BRANCH**

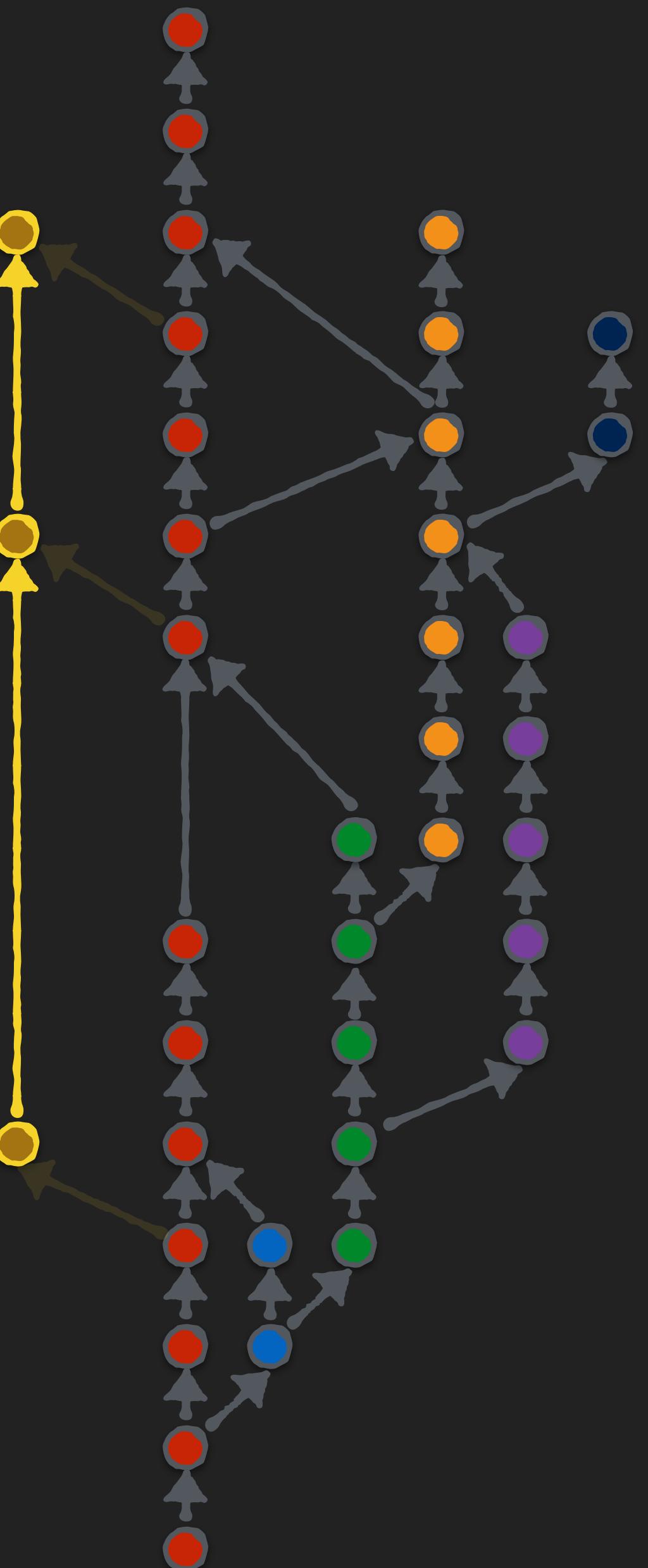
- Komplett getrennter
Versionszweig
(production → release/production)
- Wird aus dem git-Tree-Objekt
generiert
(Low-Level-Bibliothek)
- Kein Bezug zur normalen Historie,
keine Merges zum Release-
Branch



WAS BRINGT MIR DAS?

DARAUS ERGIBT SICH

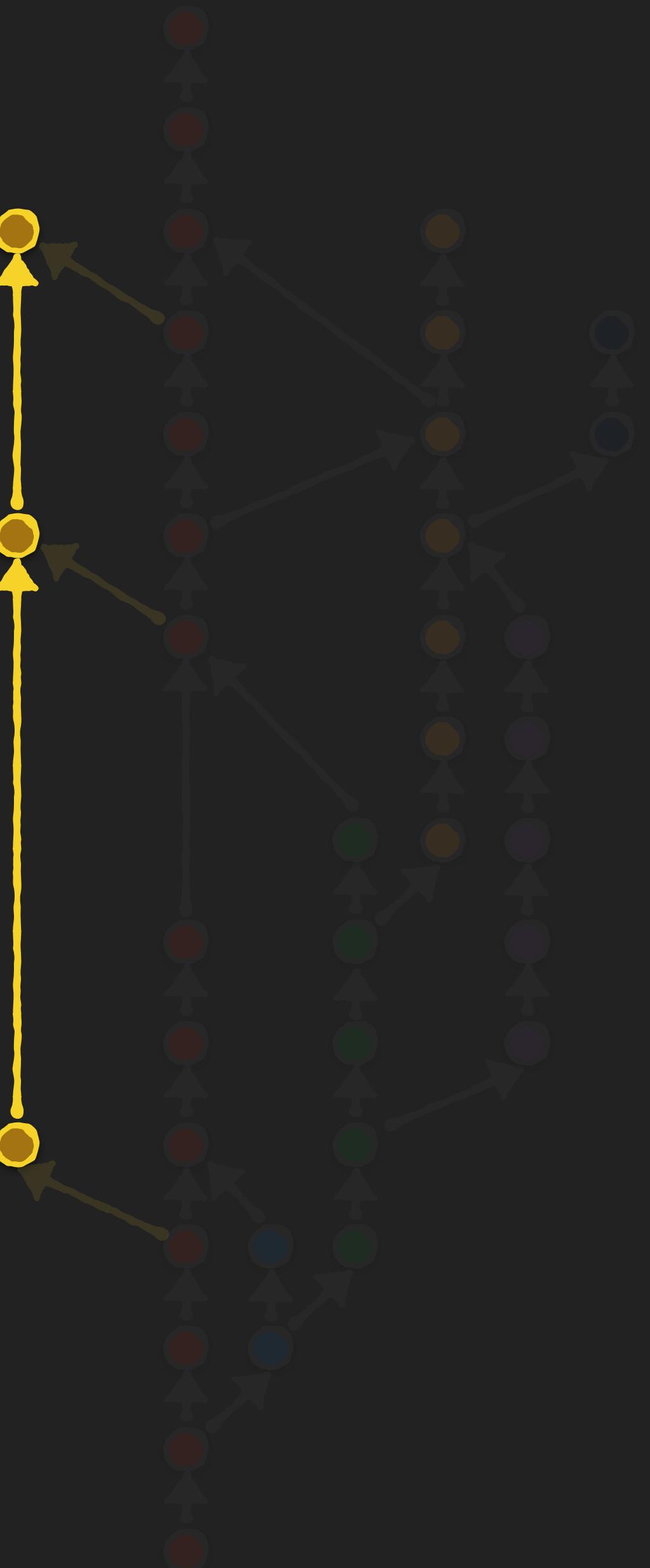
- Es können Dateien für das Deployment zum Release-Branch hinzugefügt werden (oder entfernt)
- Ermöglichung eines Build-Prozesses!
- Eigene Historie, dadurch kein Einblick in den wirklichen Entwicklungs-Ablauf
- Gleichzeitig Nachvollziehbarkeit aller Deployments für alle Entwickler



WAS BRINGT MIR DAS?

DARAUS ERGIBT SICH

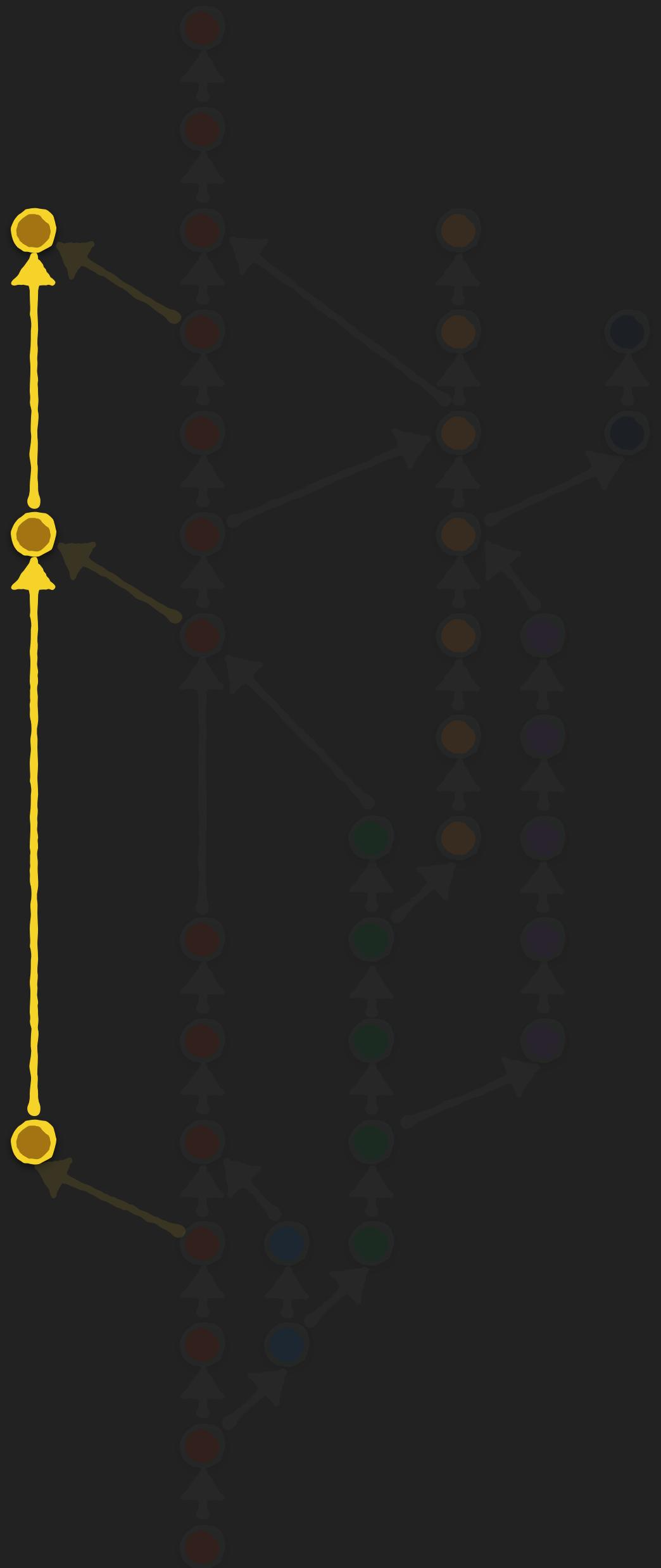
- Es können Dateien für das Deployment zum Release-Branch hinzugefügt werden (oder entfernt)
- Ermöglichung eines Build-Prozesses!
- Eigene Historie, dadurch kein Einblick in den wirklichen Entwicklungs-Ablauf
- Gleichzeitig Nachvollziehbarkeit aller Deployments für alle Entwickler

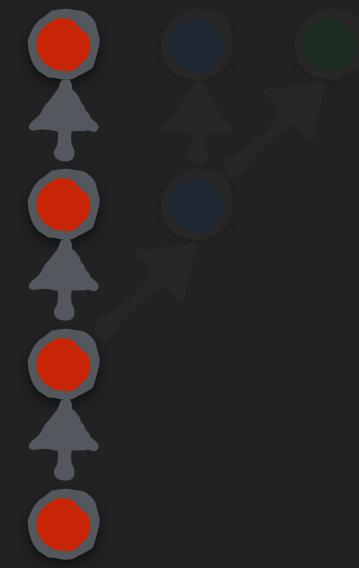


WAS BRINGT MIR DAS?

DARAUS ERGIBT SICH

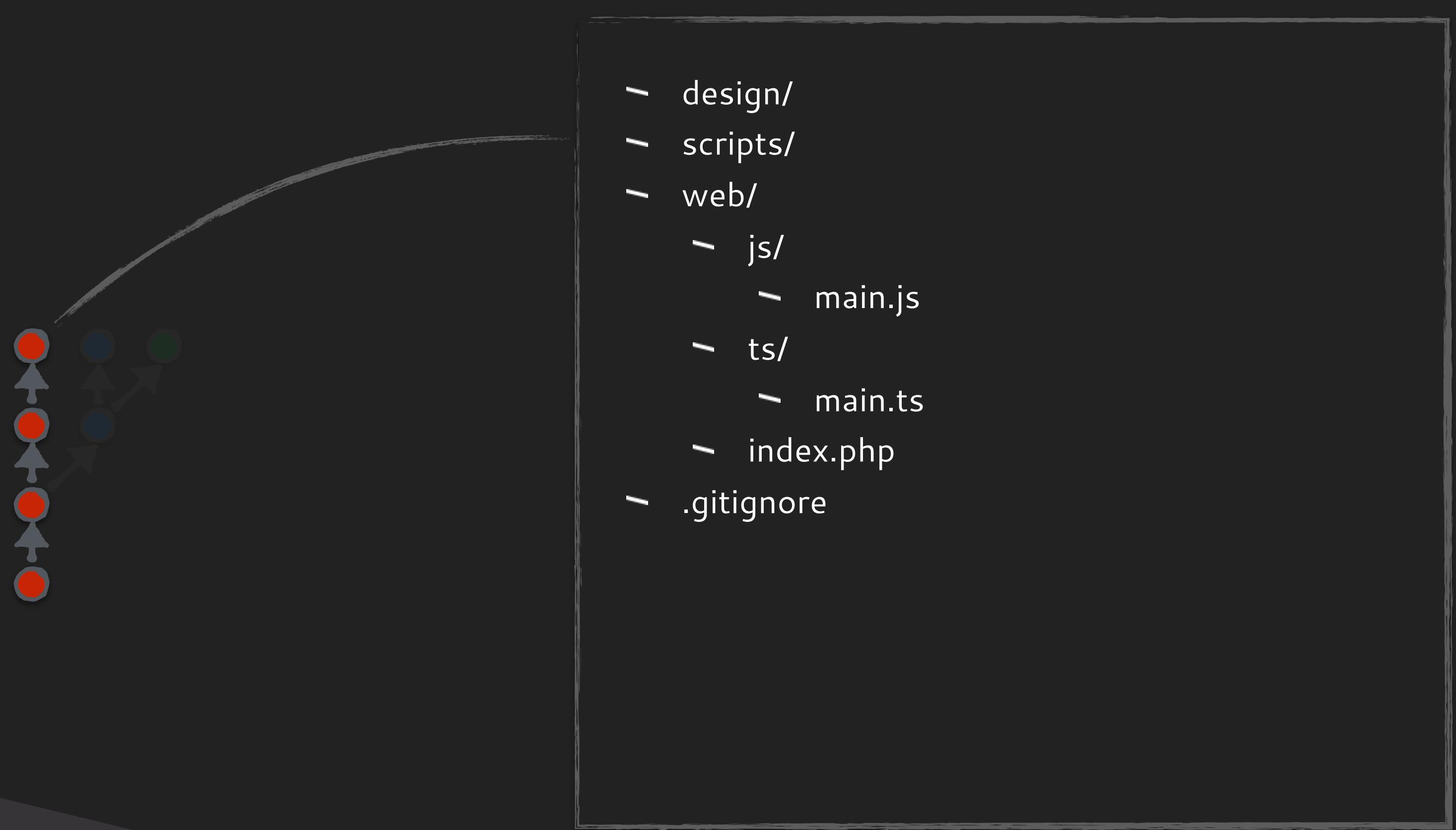
- Es können Dateien für das Deployment zum Release-Branch hinzugefügt werden (oder entfernt)
- Ermöglichung eines Build-Prozesses!
- Eigene Historie, dadurch kein Einblick in den wirklichen Entwicklungs-Ablauf
- Gleichzeitig Nachvollziehbarkeit aller Deployments für alle Entwickler
- Sehr schlanke Release-Historie
- Kann jedoch weiterhin ein normaler Branch im Repository sein
- Folgt den normalen git-Strukturen (Auch wenn normal alle Branches einen gemeinsamen Ursprung haben)





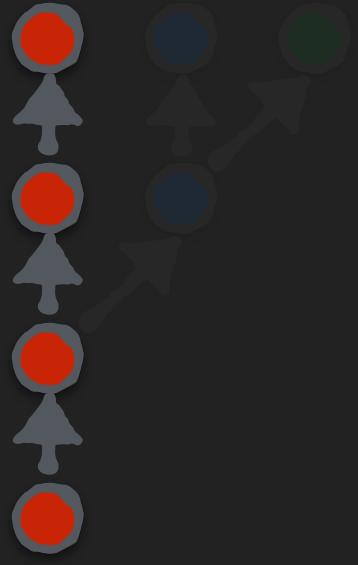
(1) LETZTER COMMIT IM RELEASE-BRANCH

IM DETAIL



(2) TREE-OBJEKT DES COMMITS LADEN

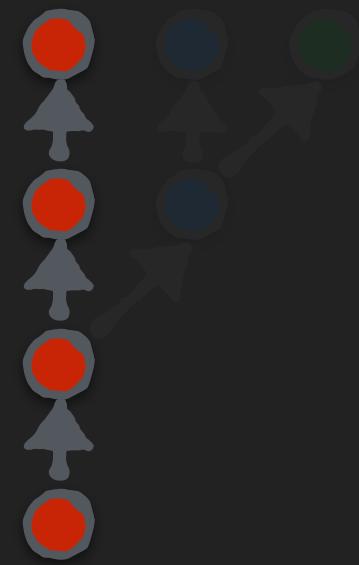
IM DETAIL



```
- design/  
- scripts/  
- web/  
  - js/  
    - main.js  
  - ts/  
    - main.ts  
  - index.php  
.gitignore
```

(3) KOPIE DES TREE-OBJEKTS ERSTELLEN

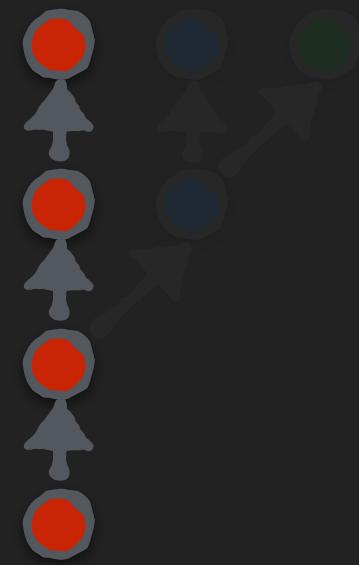
IM DETAIL



```
- design/  
- scripts/  
- web/  
  - js/  
    - main.js  
  - ts/  
    - main.ts  
  - index.php  
.gitignore
```

(4) ÜBERFLÜSSIGE ELEMENTE ENTFERNEN

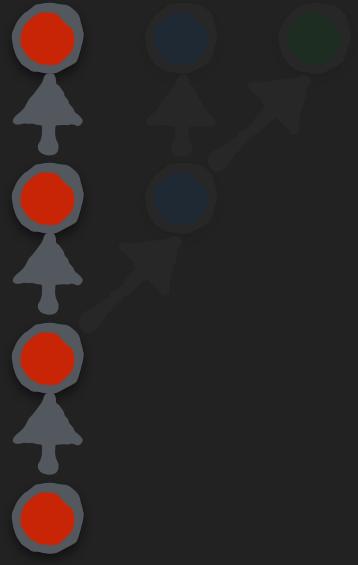
IM DETAIL



```
- design/  
- scripts/  
- web/  
  - css/  
    - style.css  
  - scss/  
    - style.scss  
  - ts/  
    - main.ts  
  - index.php  
- .gitignore
```

(5) BUILD-PROZESS – ZUSÄTZLICHE DATEIEN

IM DETAIL



```
- design/  
- scripts/  
- web/  
  - css/  
    - style.css  
  - scss/  
    - style.scss  
  - js/  
    - main.js  
  - ts/  
    - main.ts  
  - index.php  
- .gitignore
```

(5) BUILD-PROZESS – ZUSÄTZLICHE DATEIEN

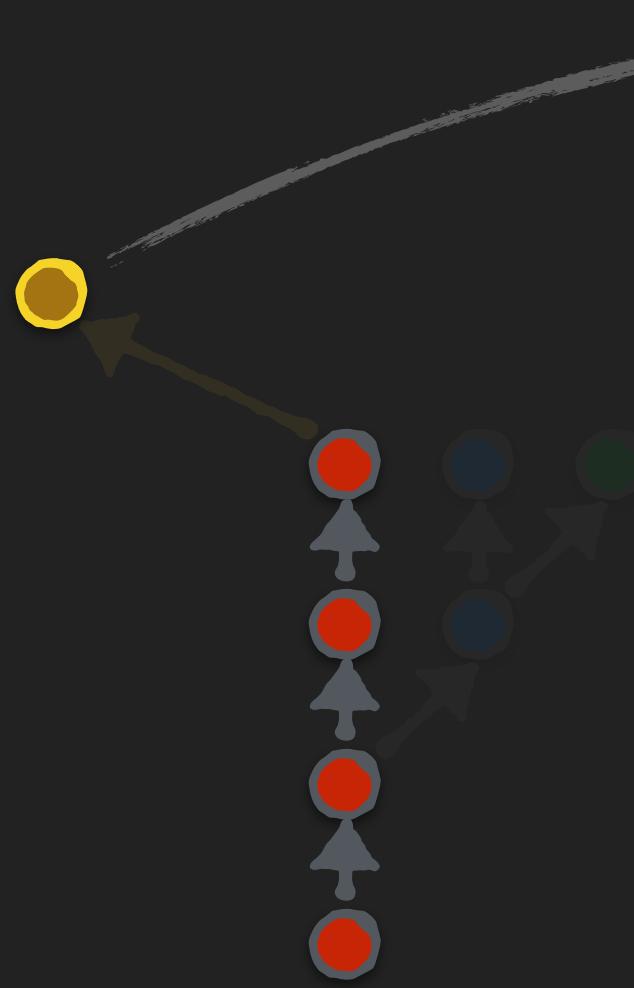
IM DETAIL



```
- design/  
- scripts/  
- web/  
  - css/  
    - style.css  
  - scss/  
    - style.scss  
  - js/  
    - main.js  
  - ts/  
    - main.ts  
  - index.php  
- .gitignore
```

(6) NEUER COMMIT MIT DIESEM TREE

IM DETAIL



```
- design/  
- scripts/  
- web/  
  - css/  
    - style.css  
  - scss/  
    - style.scss  
  - js/  
    - main.js  
  - ts/  
    - main.ts  
  - index.php  
- .gitignore
```

(6) NEUER COMMIT MIT DIESEM TREE

IM DETAIL

PRAXISBEISPIEL MIT FABDEPLOY

DRUPAL BEISPIEL

```
from fabdeploy import Git, Drupal

git = Git(local_repository_path='..', remote_repository_path="/path/to/web/",
          release_branch="production")
drupal = Drupal(drupal_path="/path/to/web/htdocs/", drush_path="/path/to/web/_drush/")

git.pull() # make sure we have the current version
git.create_release_commit() # prepare and commit new tree
git.push() # upload all changes
# git.webserver_harden_remote_git() # deny web access to .git

drupal.maintenance_enable() # enable maintenance
git.switch_release() # apply all file changes
drupal.cache_clear() # clear cache
drupal.updatedb() # apply migrations
drupal.maintenance_disable() # disable maintenance
```



DEMO

1

Kurzprofil

2

Wieso git beim Deployment?

3

Was fehlt?

4

Allgemein: git für das Deployment

5

Unser Vorgehen – git Deployment done right

6

Fazit

ZUSAMMENFASSUNG**FAZIT**

- Das Deployment sollte möglichst überall klappen – wir benötigen ein generisches Vorgehen
(Unterschiedliche Systeme, unterschiedliche Hoster)
- Durch Automatisierung lassen sich Fehler vermeiden und klare Richtlinien durchsetzen
- Standard-Jobs sollten immer ausgeführt werden
- Ein Versionsverwaltungssystem schafft Sicherheit (z.B. gegen Manipulationen)
- Es lohnt sich Out-Of-The-Box zu denken und kreativ mit den bestehenden Möglichkeiten umzugehen
- Ihr müsst selbst entscheiden, was am Besten zu eurem Anwendungsfall passt!



Vielen Dank! Fragen?

(Psst, wir suchen neue Kollegen)

TEAM23 GMBH
DAVID DANIER
WWW.TEAM23.DE