Git in großen Projekten

Einsatz von Submodulen und Subtrees

René Preißel (rp@eToSquare.de)

Buchhandlung Lehmanns, 16.10.2013

Über Mich

René Preißel (rp@eToSquare.de)
Freiberuflicher
Berater, Entwickler, Trainer

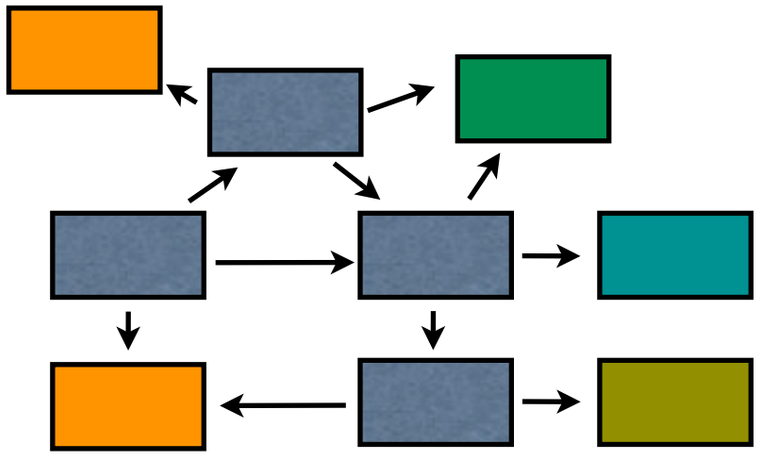
Schwerpunkte

- Software-Architekturen
- Java-Enterprise-Technologien
- **Build- und Konfigurationsmanagement**

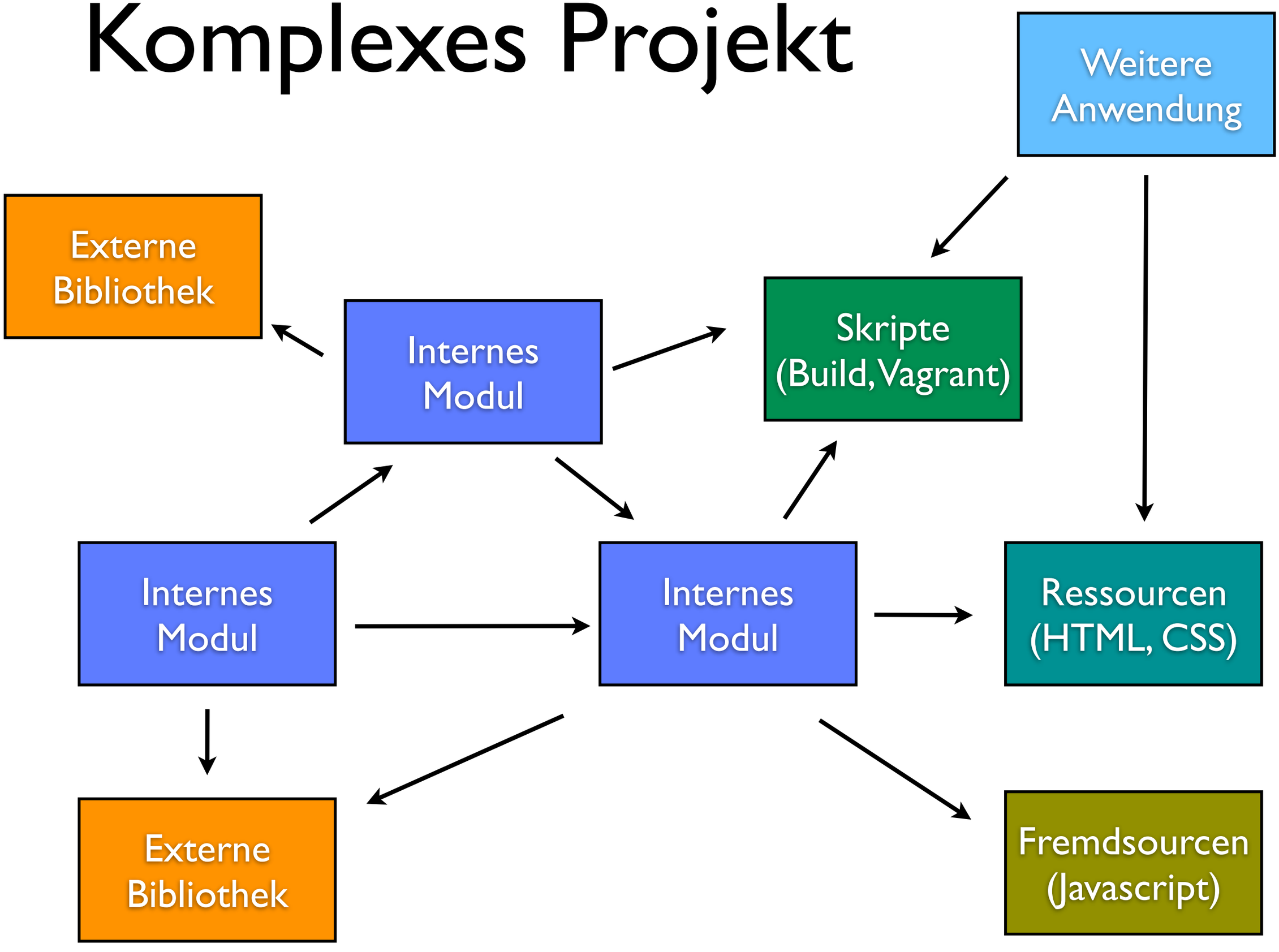
Ich unterstütze Sie bei der Einführung von Git und der Optimierung Ihrer Build-, Konfigurations- und Deployment-Prozesse (http://www.etosquare.de/git).

30 Minuten

1. Welche Abhängigkeiten gibt es in Projekten?
2. Welche Abhängigkeiten in Git verwalten?
3. Wie geht das mit Submodulen?
4. Wie geht das mit Subtrees?

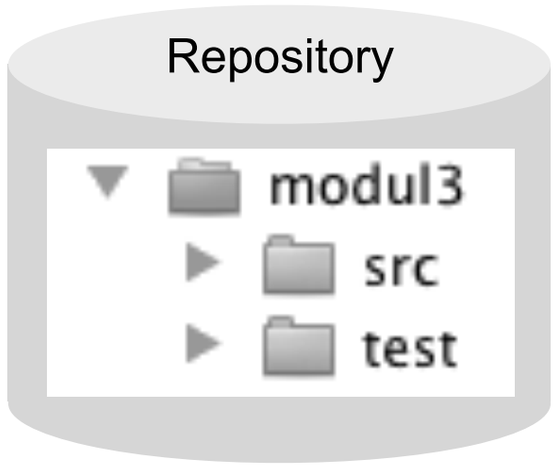


Komplexes Projekt



Modul =  Release-Einheit

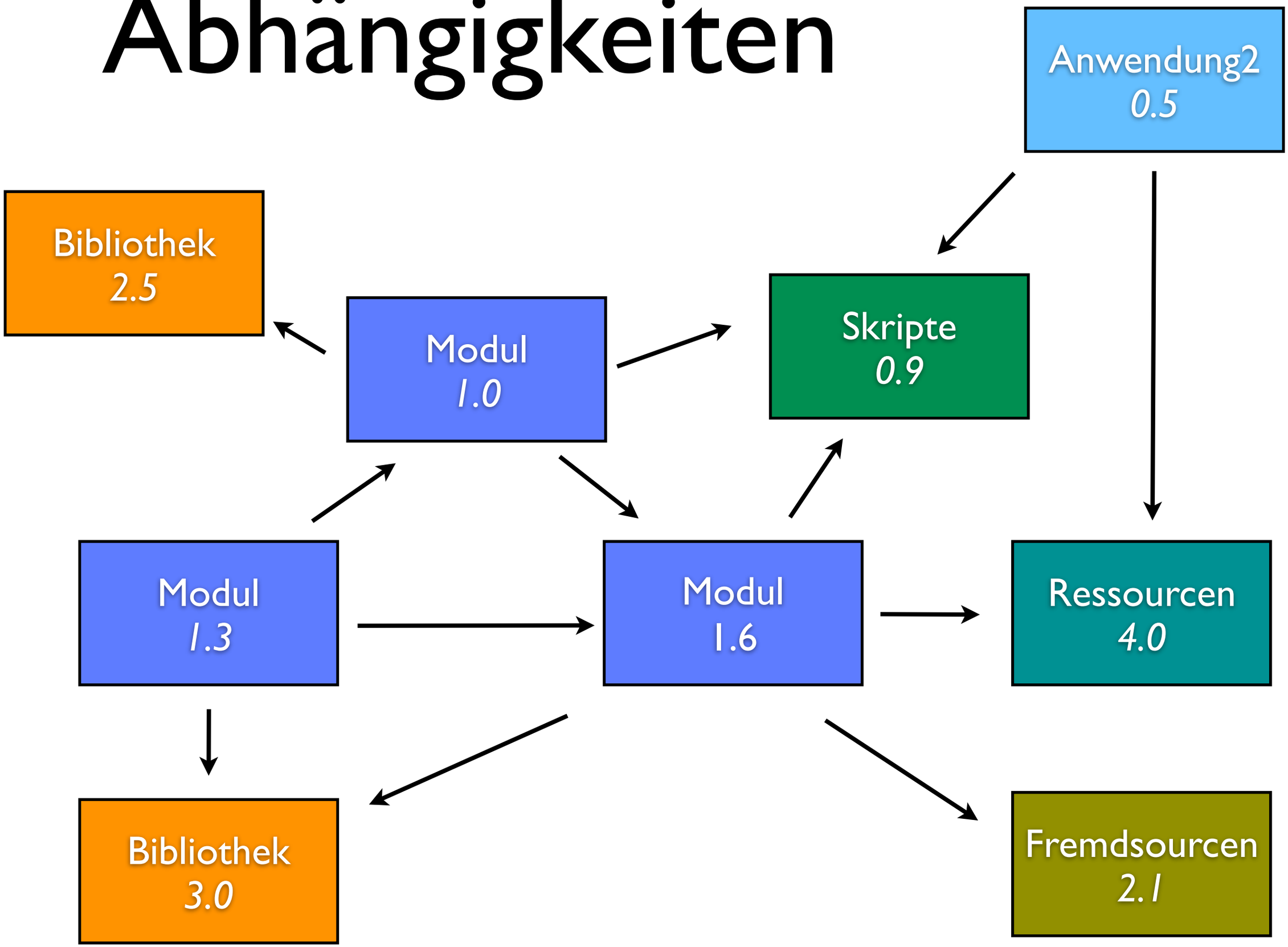
- Eigene Version
- Eigener Lebenszyklus

Modul = 

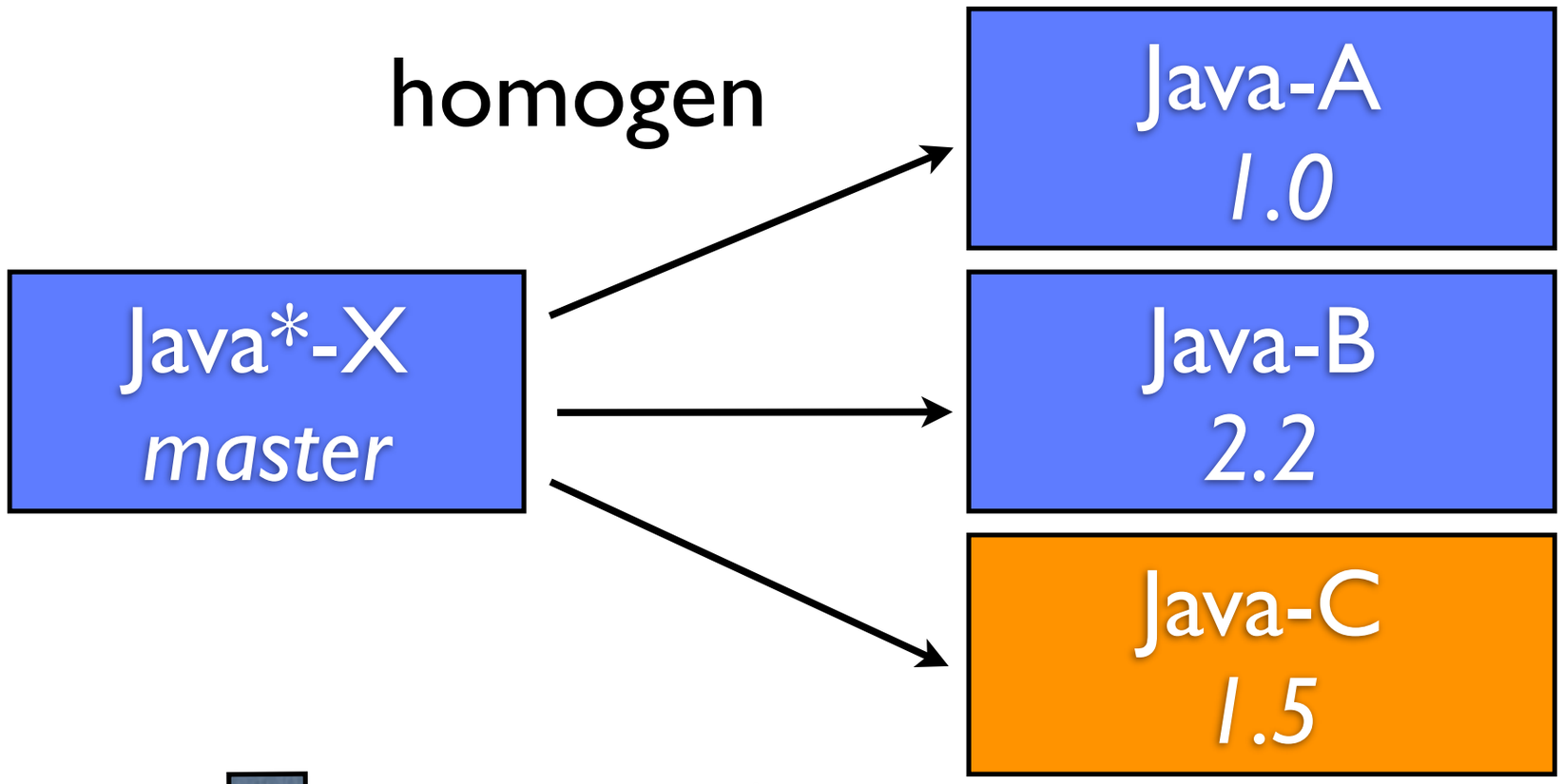
Git-Repository

- Eigene Tags und Branches

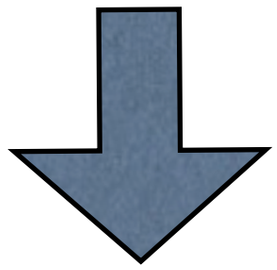
Abhängigkeiten



homogen

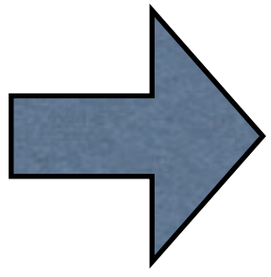


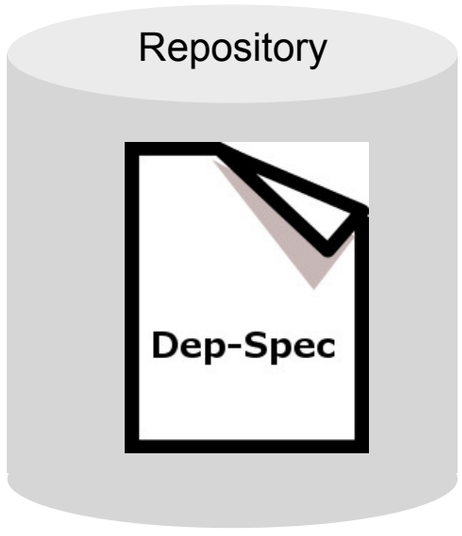
* Ersetzen mit der Technologie Ihrer Wahl

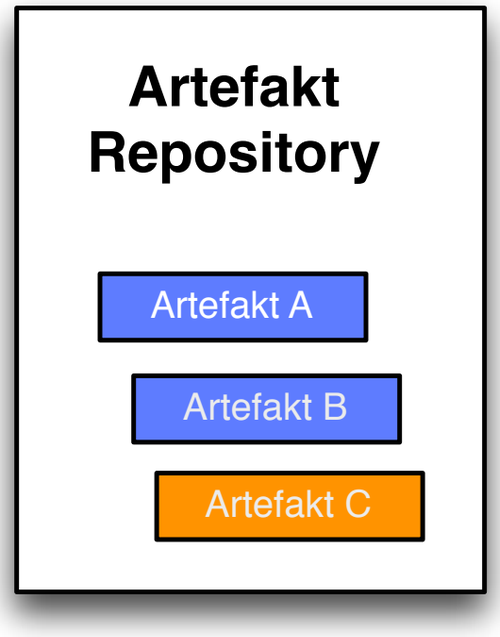


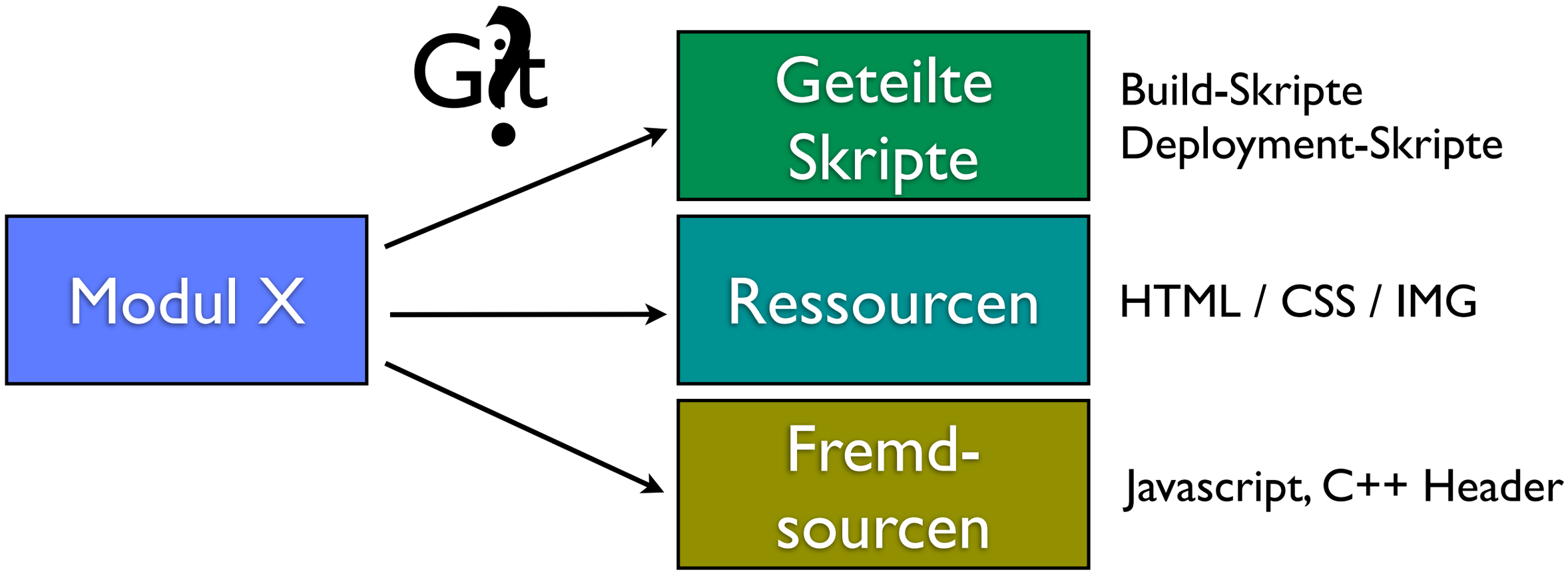
Dependency Manager

- Maven, Ivy, Gradle,
- npm, RequireJS,
- Leiningen, SBT, ...

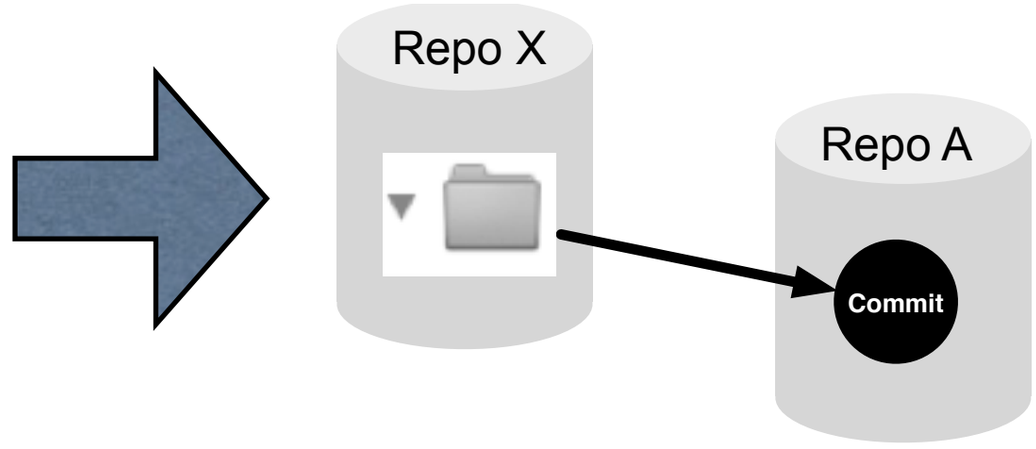






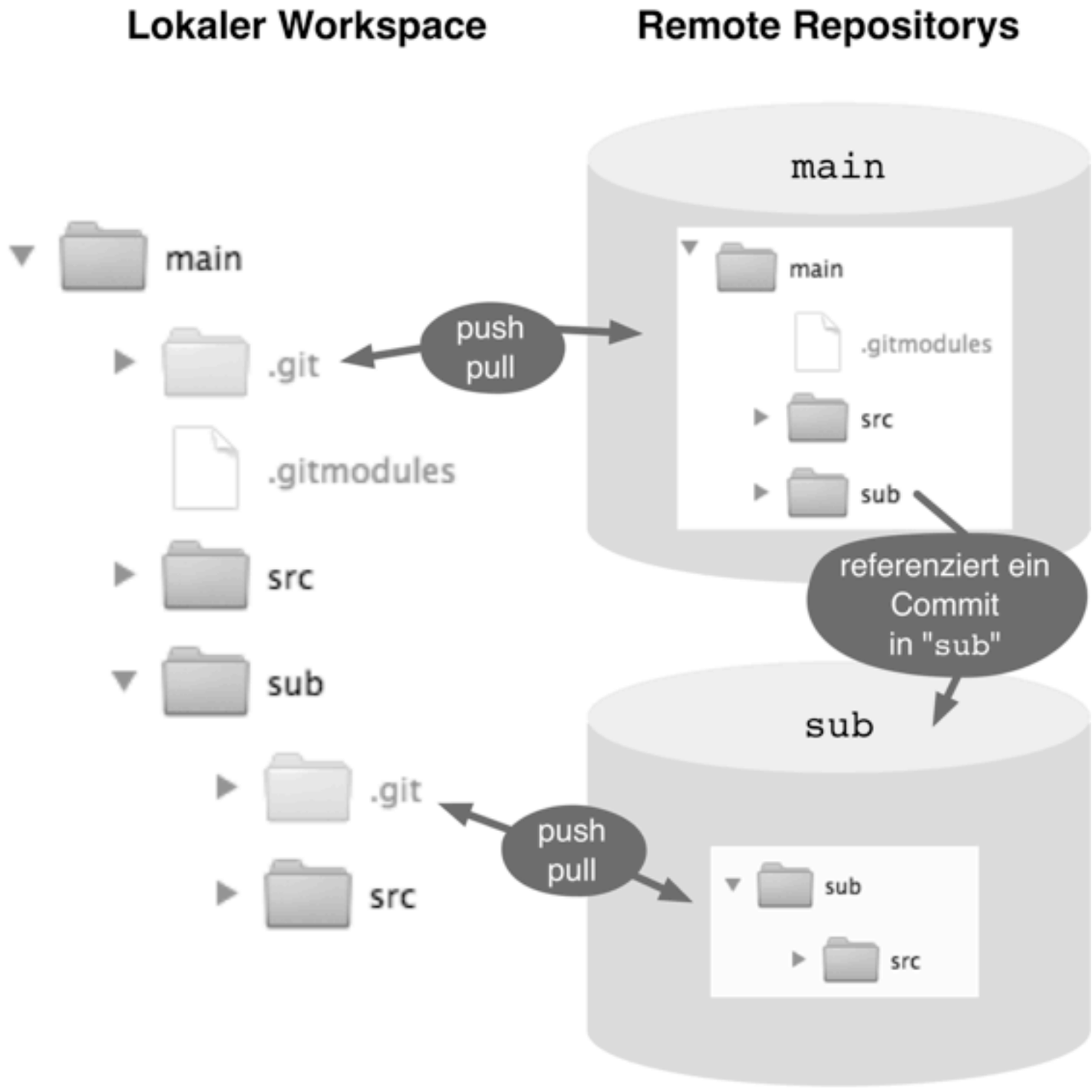


- Inhomogene Infrastruktur
- Einbindung von Sourcen und Ressourcen
- Globales Build erforderlich



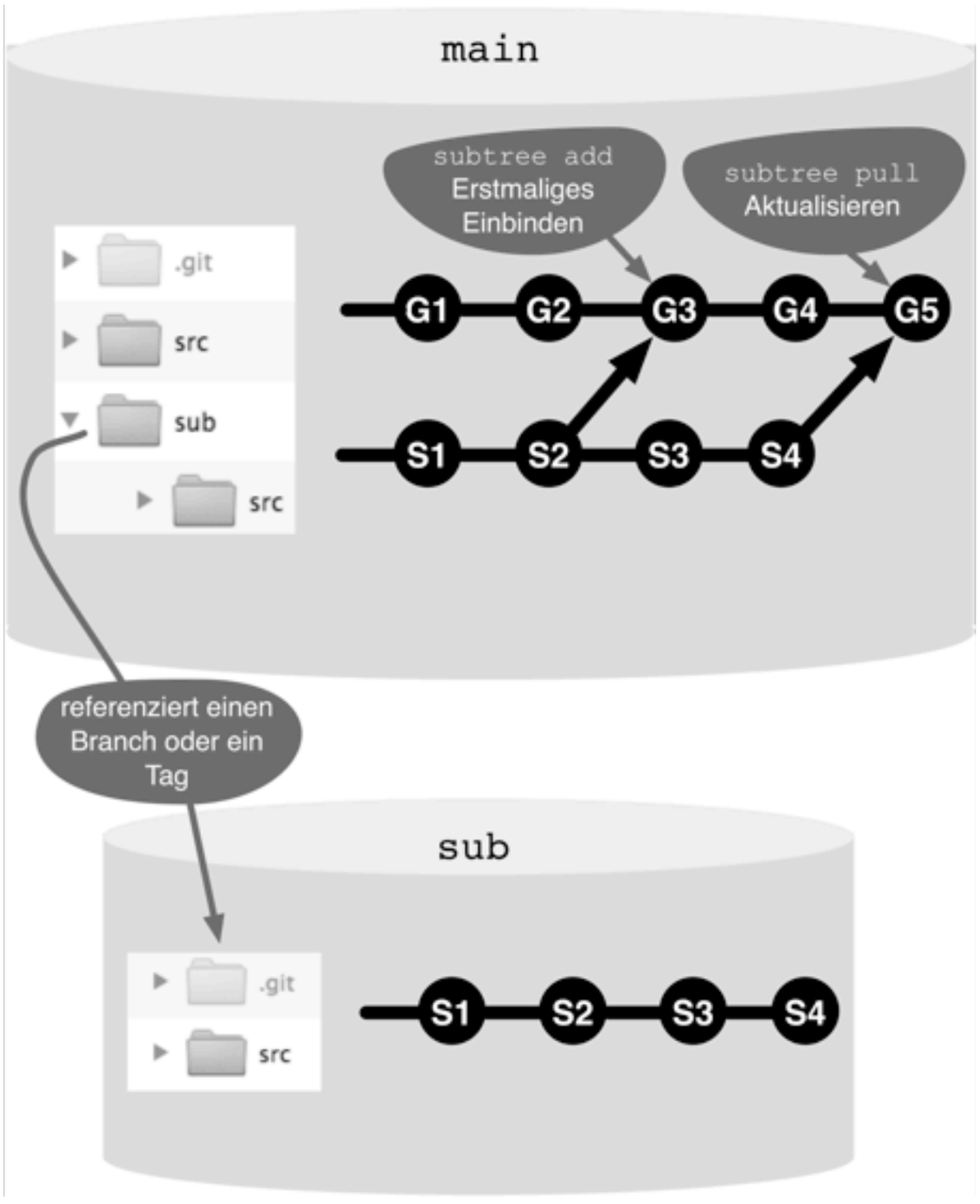
Submodule oder Subtrees

Submodule



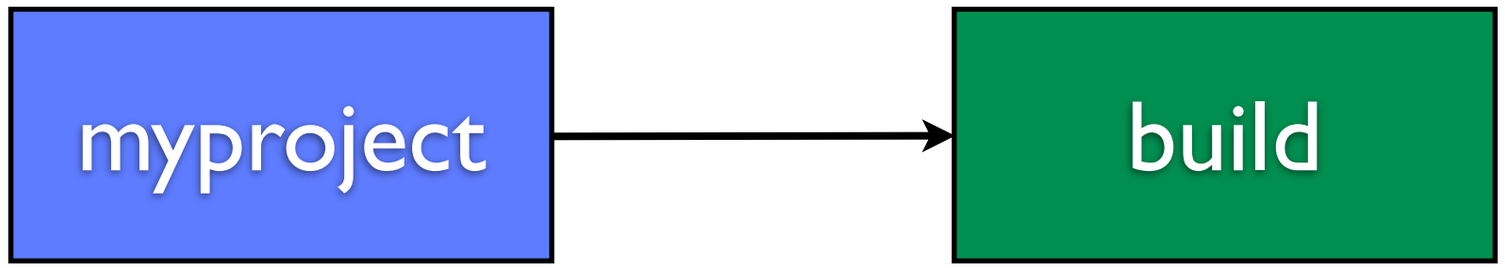
* aus „Git - Grundlagen und Workflows“

Subtree



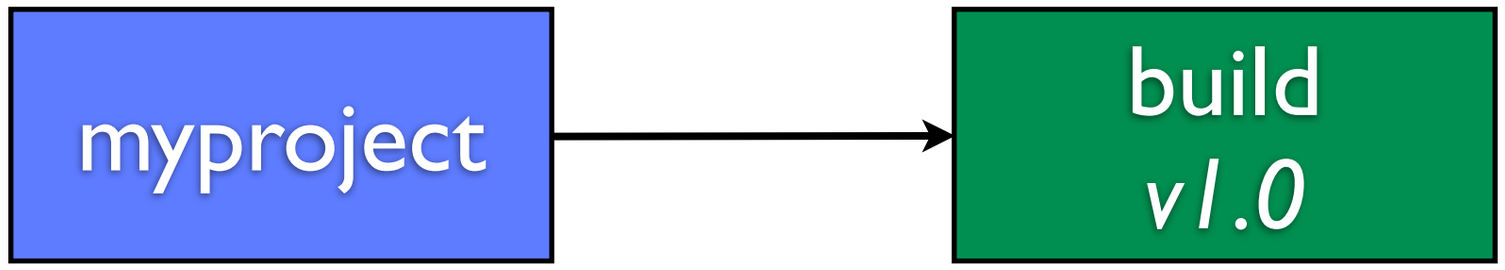
* aus „Git - Grundlagen und Workflows“

Beispiel

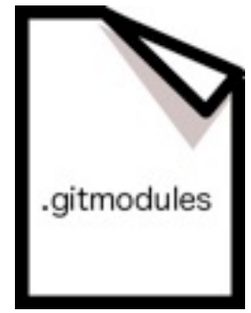


- Schritte:
  - Modul hinzufügen
  - Repository mit Modulen klonen
  - Neue Version eines Moduls einbinden
  - Änderungen in einem Modul durchführen

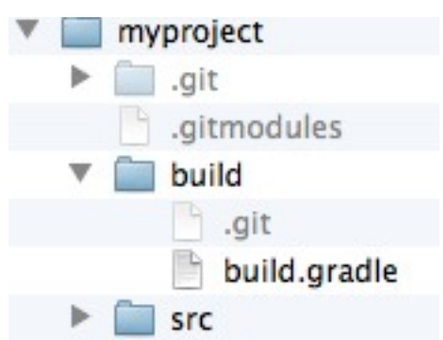
Submodul hinzufügen




```
myproject$ git submodule add ../build.git build
myproject$ cd build
myproject/build$ git checkout v1.0
myproject/build$ cd ..
myproject$ git commit -m “add Submodul“

[master cbe9e65] add submodule
 2 files changed, 4 insertions(+)
create mode 100644 .gitmodules
create mode 160000 build

myproject$ git push
```

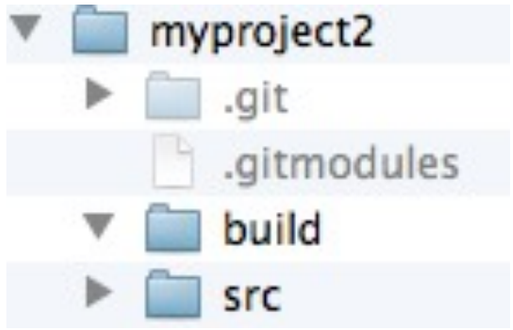



```
[submodule "build"]
    path = build
    url = ../build.git
```

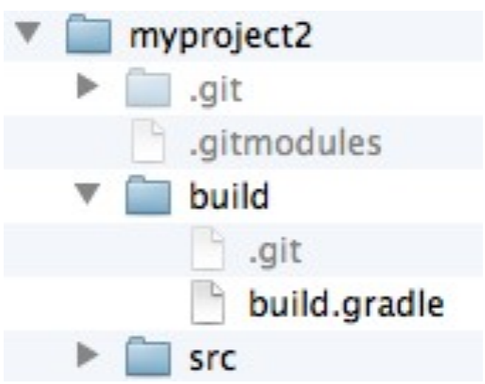


Klonen mit Submodulen

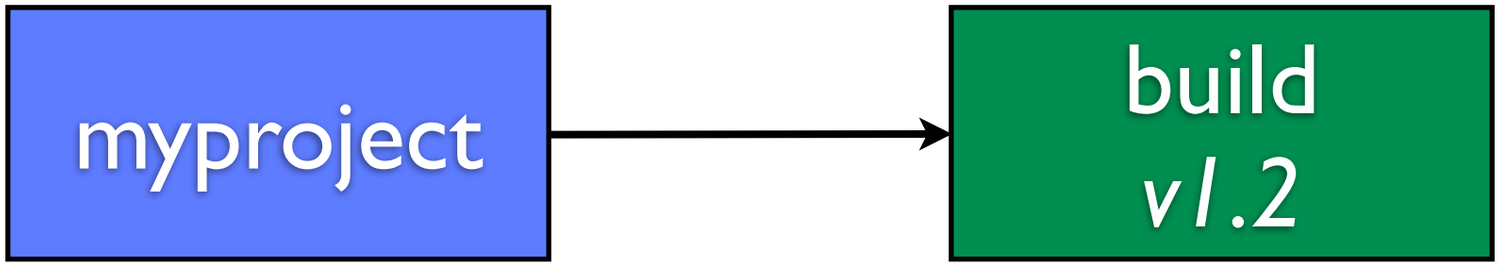
```
projects$ git clone myproject.git myproject2
projects$ cd myproject2

#.gitmodules in .git/config übertragen
myproject2$ git submodule init

#Repositories der Submodule klonen
myproject2$ git submodule update


#Ein Schritt
myproject2$ git submodule update --init
```



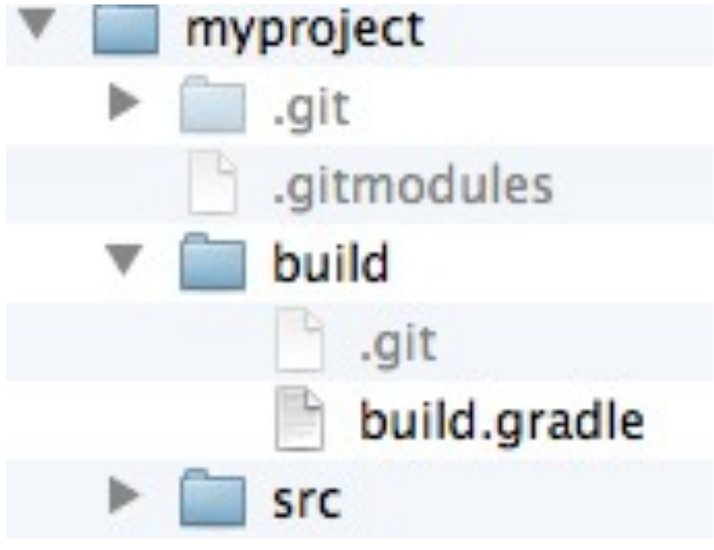

```
#.git/config
[submodule "build"]
    url = ../build.git
```



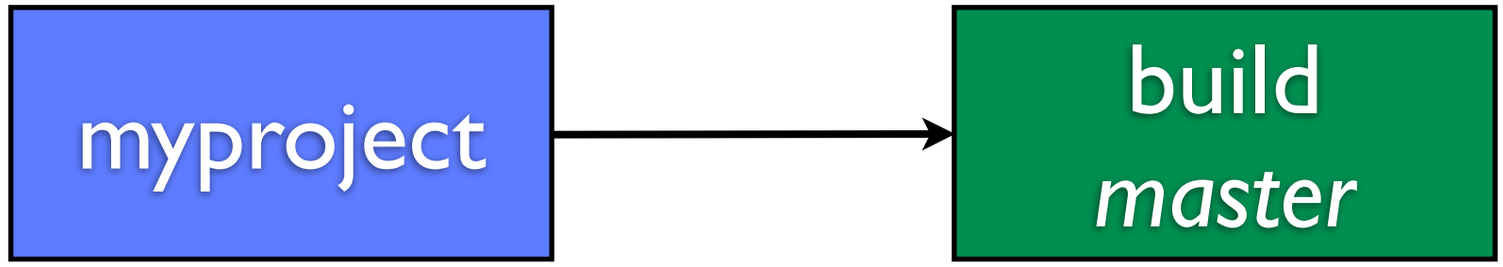
Neue Version festlegen




```
myproject$ cd build
myproject/build$ git fetch
myproject/build$ git checkout v1.2
myproject$ cd ..
myproject$ git add build
myproject$ git commit -m "neue Version"
myproject$ git push

#Im anderen Repo aktualisieren
myproject2$ git pull
myproject2$ git submodule update --init
```



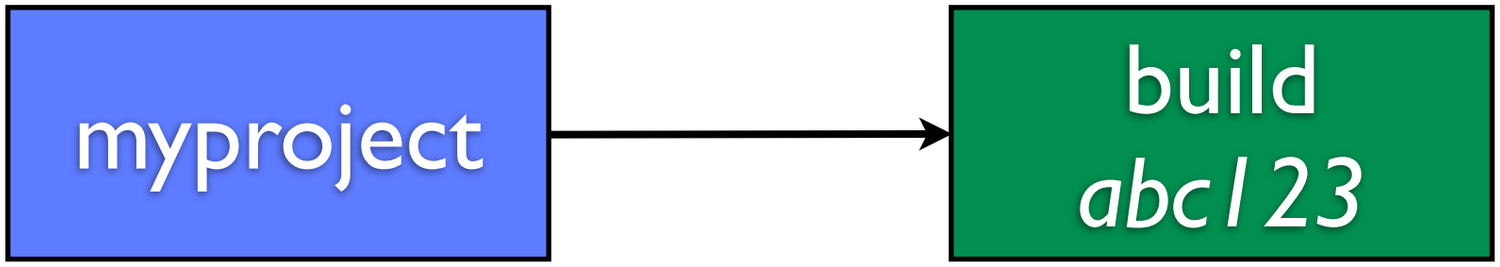
Aktuellste Version holen



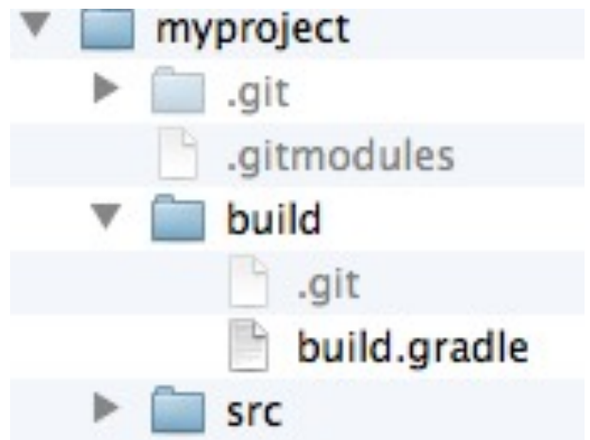

```
#Branch-Eintrag in .gitmodules beim add (seit 1.8.2)
myproject$ git submodule add --branch master ../build.git build

#--remote holt den aktuellen Head des konfigurierten Branches
myproject$ git submodule update --remote
myproject$ git add build   #ggf. noch andere Submodule
myproject$ git commit -m "neue Version"
myproject$ git push
```

```
[submodule "build"]
    path = build
    url = ../build.git
    branch = master
```

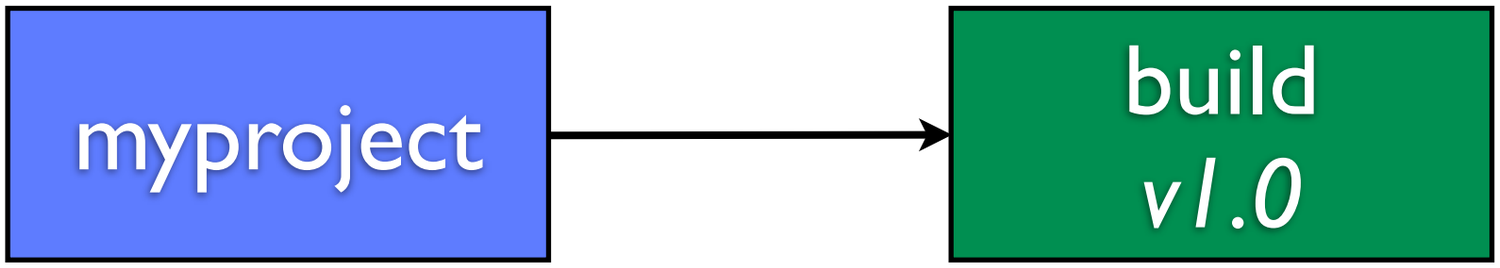
In Submodulen arbeiten




```
myproject$ cd build
myproject/build$ git fetch
myproject/build$ git checkout master
#Dateien ändern
myproject/build$ git commit -am "neue Version in Submodul"
myproject$ cd ..
myproject$ git add build
myproject$ git commit -m "neue Version"
myproject$ git push --recurse-submodules=on-demand

#Alias pushall
$ git config --global
        alias.pushall 'push --recurse-submodules=on-demand'
```

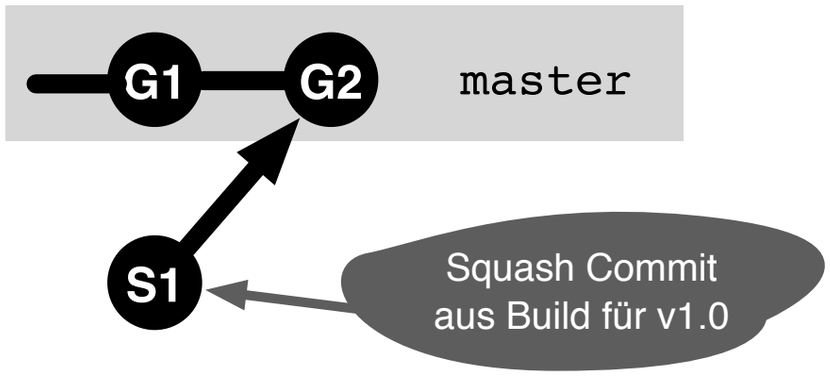


Subtree hinzufügen

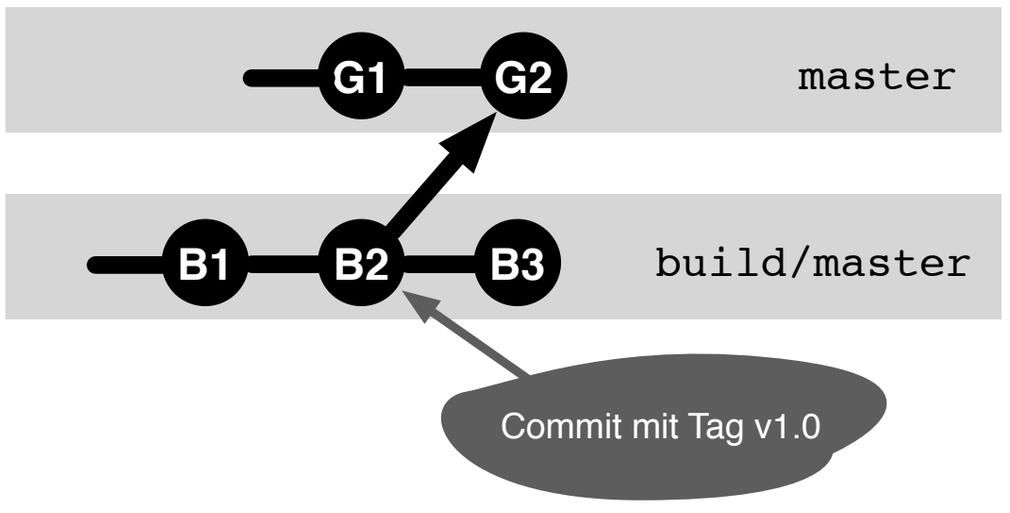


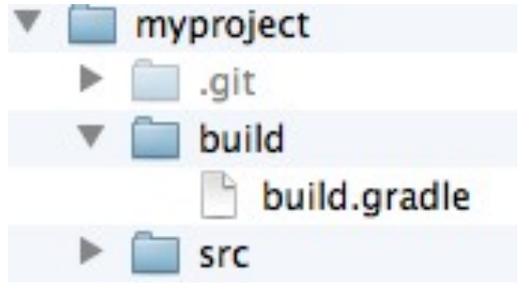

```
myproject$ git subtree add --prefix build --squash ../build.git v1.0
myproject$ git push
```

mit --squash



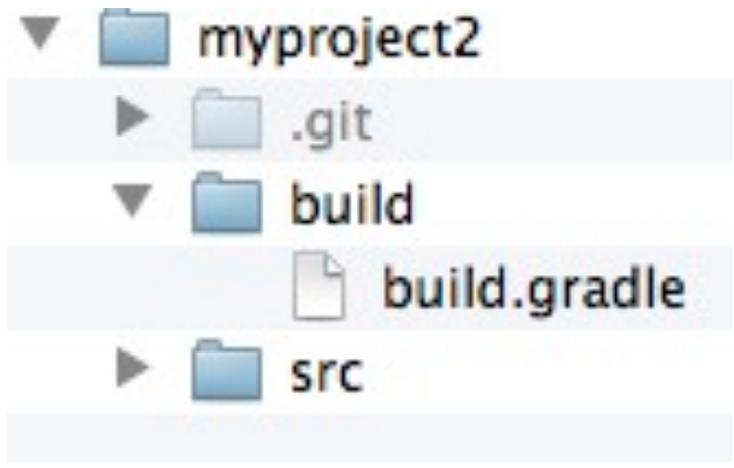
ohne --squash



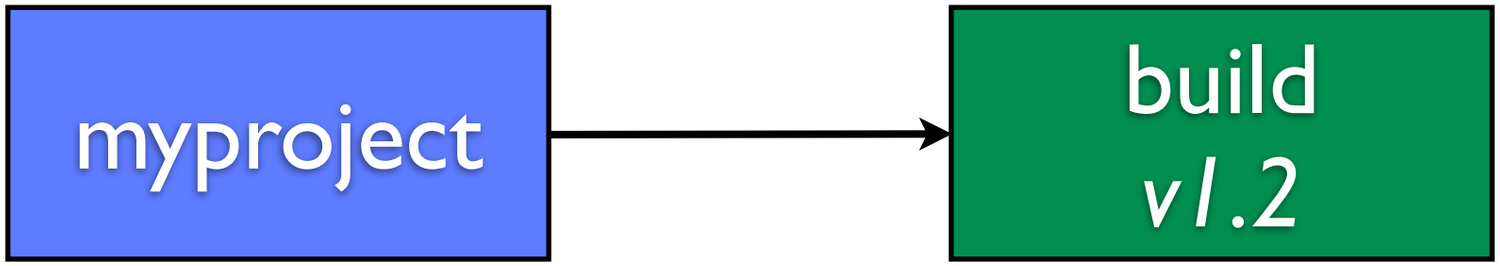


Klonen mit Subtrees

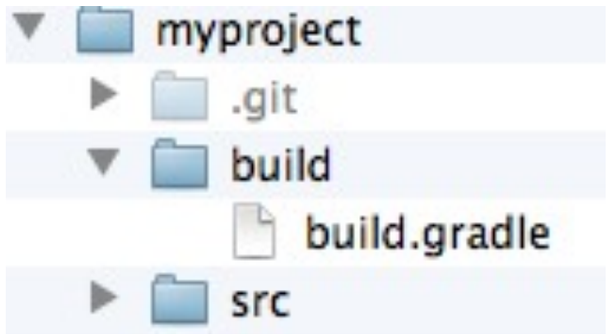
```
projects$ git clone myproject.git myproject2
```



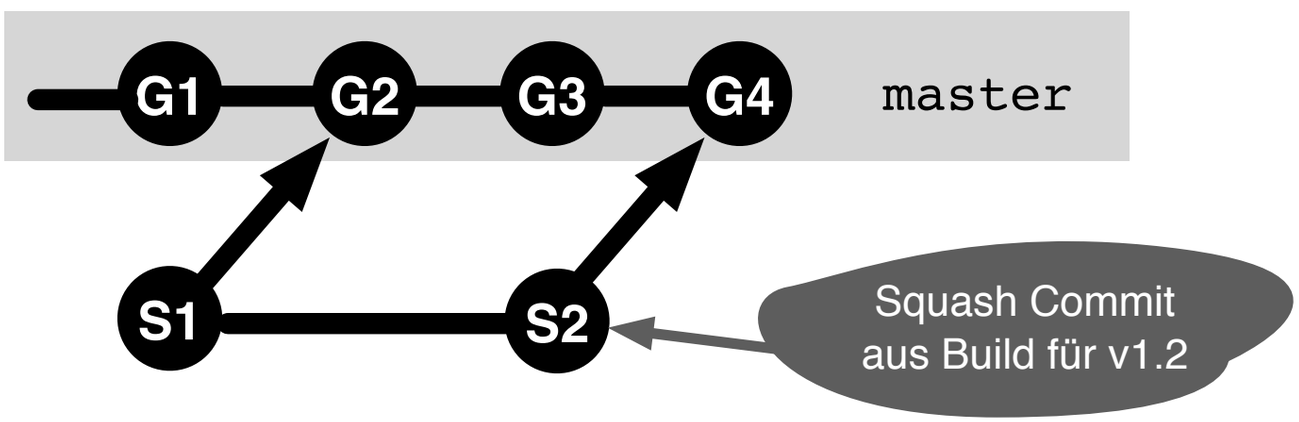
Neue Version festlegen



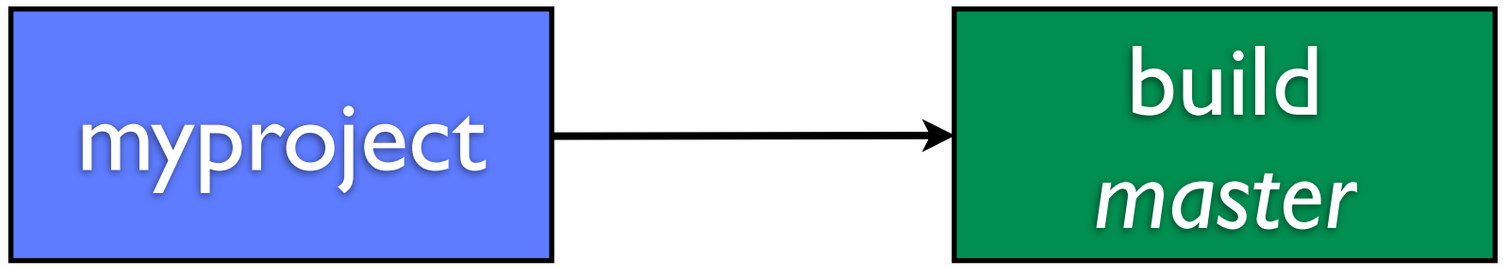

```
myproject$ git subtree pull --prefix build --squash ../build.git v1.2
myproject$ git push

#Im anderen Repo aktualisieren
myproject2$ git pull
```



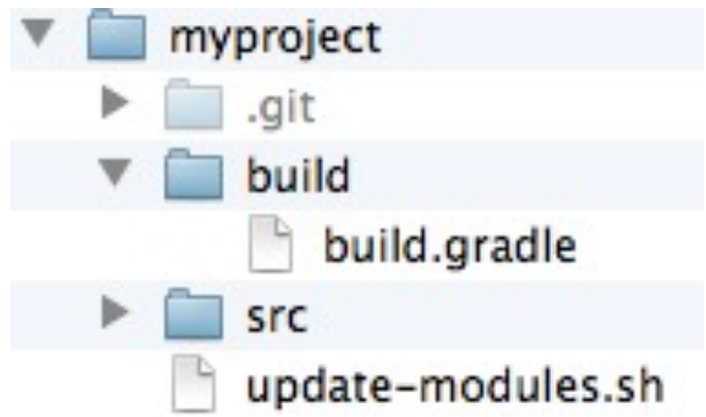
mit --squash



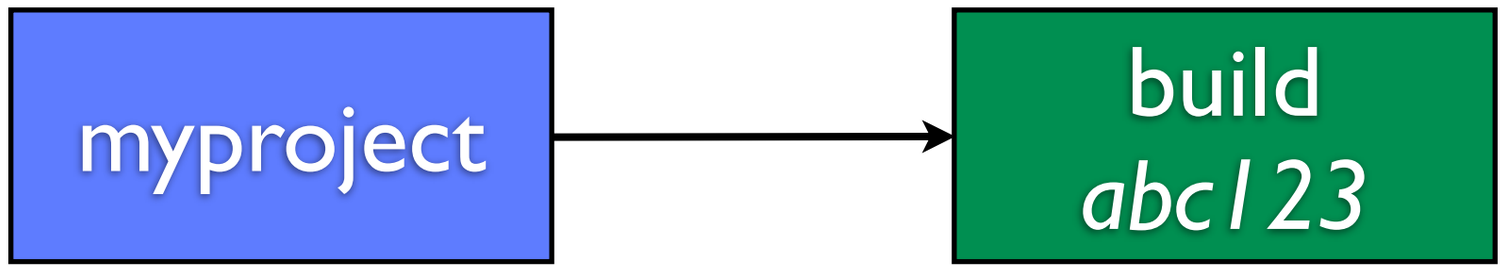
Aktuellste Version holen



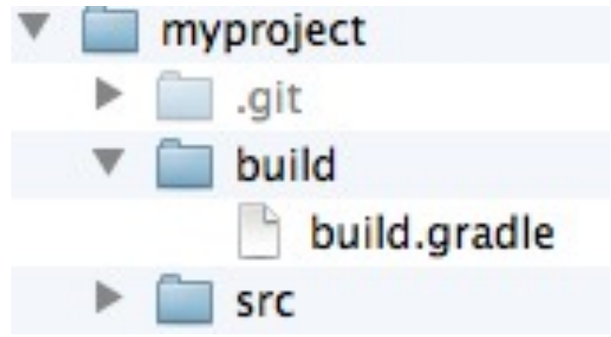

```
myproject$ git subtree pull --prefix build --squash ../build.git master
myproject$ git push

#Eigenes Update-Skript anlegen: update-modules.sh
#Remote verwenden um Pfade im Skript zu vermeiden
myproject$ git remote add build ../build.git

myproject$ ./update-modules.sh
```

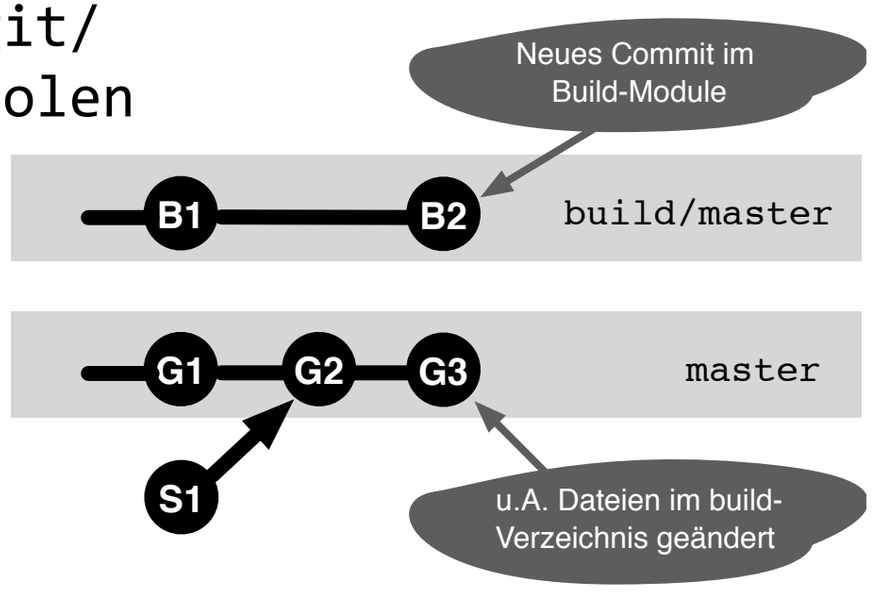


In Subtrees arbeiten (I)

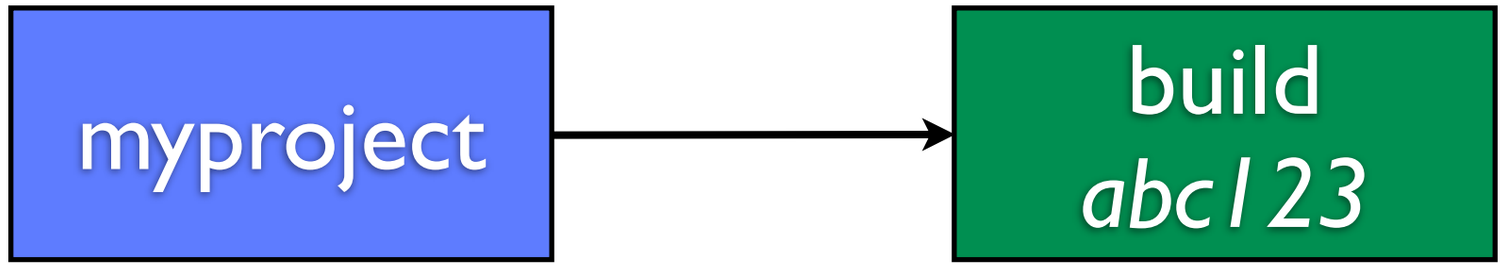



```
#Dateien im build-Verzeichnis ändern
myproject$ git commit -am "neue Version in Subtree"
myproject$ git push

#Änderungen in das Modul zurückübertragen
#Einmalig Remote anlegen
myproject$ git remote add -f build ../build.git/
myproject$ git fetch build   #letzten Stand holen
```

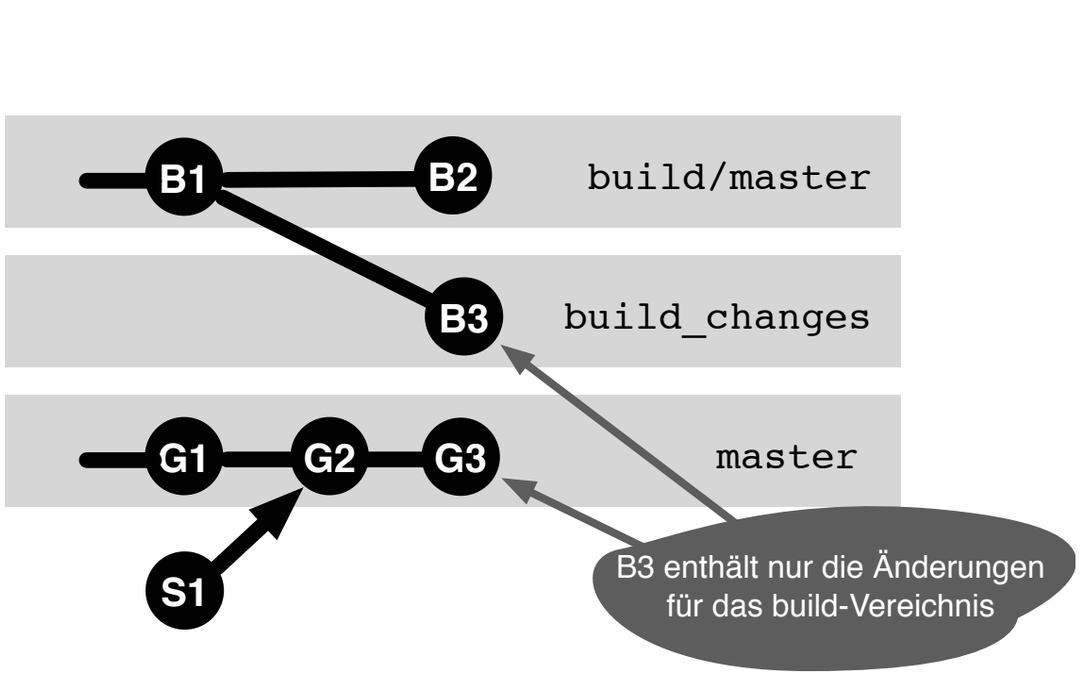


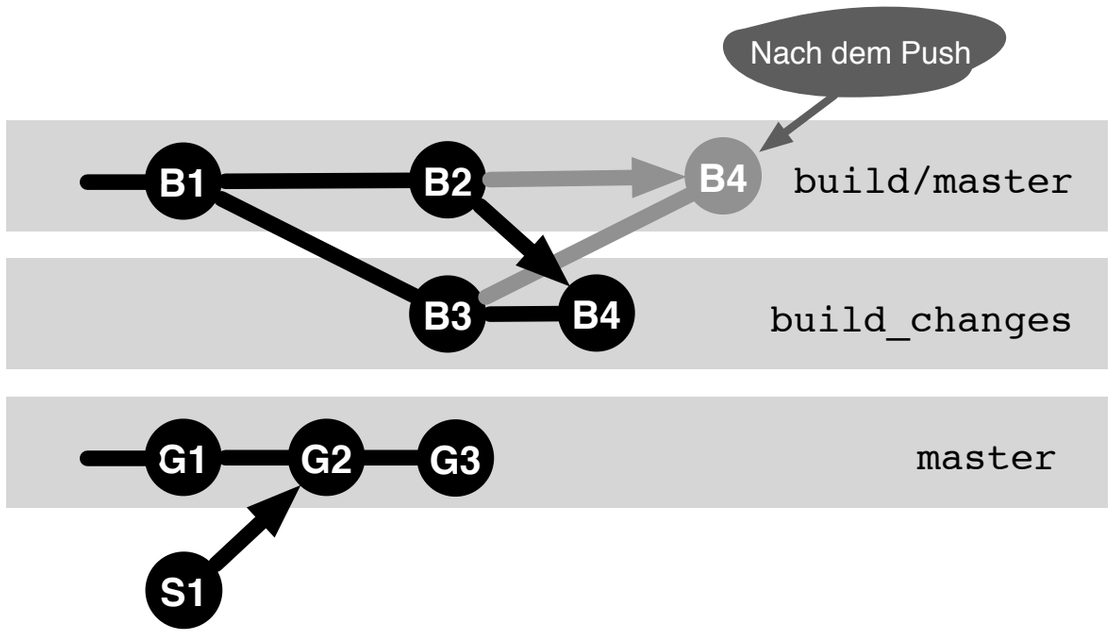


In Subtrees arbeiten (II)




```
myproject$ git subtree split --prefix build --branch build_changes
myproject$ git checkout build_changes
myproject$ git merge build/master  #mit lokaler Version vereinigen
myproject$ git push build HEAD:master #Zurück in das originale Repo
```





Subtree vs. Submodulen

- Die Verwendung von Subtrees ist meistens weniger komplex als die Verwendung von Submodulen.
- Die Komplexität trifft nur denjenigen, der auch direkt mit den Modulen arbeitet.
- Ein Repository enthält alle Dateien.
- Bei Subtrees ist die Historie der Dateien nicht im Hauptrepository einsehbar (--squash).

Schlußwort

1. Überprüfen ob Module wirklich einen eigenen Releasezyklus benötigen.
2. Am besten Abhängigkeiten von Modulen mit externen Dependency-Manager verwalten.
3. Subtrees bevorzugen.