Git

Basiskonfiguration git

Es bietet sich an in seinem Home Verzeichnis (Linux: /home /<username>) eine Datei '.gitconfig' anzulegen. Meine Datei sieht wie folgt aus:

Ich empfehle *jedem*, diese Datei entsprechend seiner Vorlieben anzupassen (insbesondere push.default). (Eine vollständige Übersicht darüber, was man da alles konfigurieren könnte liefert 'man git config').

Achtung Windows-Benutzer: Die Einstellung autocrlf = true führt dazu, dass exportierte Nuclet-Dateien die Quellcode enthalten (Datenquellen, Regeln) auch den Windows-Zeilenumbruch CRLF besitzen. Das kann beim Import des Nuclets auf Nicht-Windows-Systemen zu Problemen führen.

```
[user]
        name = <mein_name>
        email = <mein_name>@gmx.de
                # signingkey =
8C4D5A98

[diff]
        color = auto

[page]
        color = true

[status]
        color = auto

[core]
        editor = kate
        safecrlf = true
        # Unix/Mac only
        autocrlf = input
        ## Windows only
        # autocrlf = true

                ## Windows only, if Error Message:
"Filename too long"
                ## CMD: git config --system core.
longpaths true
                # longpaths = true
 [merge]
        tool = kdiff3

[color]
        ui = auto
[push]
        # defines what 'git push' means, see 'man
git-config' section push.default for details
        # possible values are: nothing, matching,
upstream, simple, current
        # default = matching # default for git 1.x
but only recommented for experienced git users
        default = simple # future default for git
2.x

[gpg]
                program = gpg2

[branch]
                autosetuprebase = always

[rerere]
                enable = true
                autoupdate = true
```

Dokumentation und GUI/Clients

- Die ultimative Referenz kann online und auf deutsch hier gelesen werden.
- Online stehen auch die Manpages zur Verfügung.
- Im Git Wiki befinden sich ebenfalls viele interessante Inhalte, z. B. Links zu weiterer Dokumentation und eine Liste von GUI git Clients.
- egit, der Eclipse/STS git team support, besitzt ebenfalls eine ausführliche Dokumentation.
- Die Weboberfläche von Bitbucket hat ebenfalls einige spezielle Feature: Get Started und Dokumentation.
- Eine alternative Benutzeroberfläche (GUI) für Mac Osx ist SourceTree. Maik mag sie und ihre Dokumentation.
- Für Linux ist (das mit git gebündelte) gitk der GUI Client der Wahl. Ferner gibt es das graphische Commit Tool git gui.
- Ein guter GUI Client ist ferner SmartGit. SmartGit läuft auf allen Betriebssystemen, aber für den kommerziellen Einsatz wird eine kostenpflichtige Lizenz benötigt. (Daher ungeeignet für die Hauptberuflichen Nuclos Entwickler.)

Erstmaliges Auschecken eines Branches

Hat man aus bitbucket.org zum ersten mal ausgecheckt, dann hat man zwar alle 'remote Branches' (git branch -r), aber nur einen lokalen 'tracking branch' (git branch), nämlich den der 'remotes/origin/master' folgt: 'heads/master' oder kurz 'master'.

Die anderen 'tracking branches' müssen also noch erstellt werden! In eclipse/sts geht das wie folgt:

Project Explorer -> nuclos (Hauptprojekt) -> rechte Maustaste -> Team -> Switch to -> New branch ...

Hier wird also der richtige 'remote branch' gewählt. Eclipse/STS schlägt dann *automatisch* einen Namen für den lokalen 'tracking branch' vor. Alternativ funktioniert's auch auf der Kommandozeile mit 'git branch -l -t 3.8 origin/3.8'.

> Sollte hier der gewünschte Branch nicht angezeigt werden, dann ist das lokale Repository wahrscheinlich nicht aktuell genug. In diesem Fall hilft ein 'git fetch' bzw. in eclipse/sts: Project Explorer -> nuclos (Hauptprojekt) -> rechte Maustaste -> Team -> Fetch from Upstream.
>
> Ein *Fetch* ist übrigens immer zum Abgleich eine gute Idee - und er macht nie etwas kaputt, da er den gerade ausgecheckten Stand ('working copy' und 'index') nicht verändert.

Zeilenende Policy

In einem neu ausgecheckten Repository bitte gleich die Zeilenende Policy richtig einstellen. Nuclos verwendet Standardmäßig LF beim Export eines Nuclets (ab v4.25 Siehe auch NUCLOS-6629). Damit wird eine Betriebssystemübergreifende Zusammenarbeit ermöglicht, und sollte daher von einem GIT Client nicht automatisch "korrigiert" werden!

Windows/Linux/Mac:

```
cd <repo>
git config core.autocrlf input
git config core.safecrlf true
```

Rebase als 'pull strategy'

Sehr wichtig ist es, als 'Pull stategy' *Rebase* auszuwählen (wie im Screenshot geschehen), da so unnötige 'Merges' durch Pull vermieden werden! Genauere Informationen hierzu finden sich unter [http://git-scm.com/book/de/Git-Branching-Rebasing] und [http://www.jarrodspillers.com/2009/08/19/git-merge-vs-git-rebase-avoiding-rebase-hell/] (beides *Pflicht lektüre* für Entwickler!).

Diese Erstellung des 'tracking branch' muss nur *einmal* durchgeführt werden. Danach finden man die schon erstellten 'tracking branches' in Eclipse/STS direkt im 'Switch to' Menü:

> Nach dem Auschecken ist die letzte Zeile 'rebase = true' für den Branch 'master' *nicht* gesetzt. Sie sollte aber gesetzt sein. ***Alle* Nuclos-internen Entwickler fügen daher diese Zeile in der Datei <repository>/.git/config hinzu.** Der Abschnitt sollte dann insgesamt wie folgt aussehen:
>
> [branch "master"]
> remote = origin
> merge = refs/heads/master
> **rebase = true**
>
> Die Zeile kann mit einem einfachen Texteditor hinzugefügt werden. Noch einfacher ist es mit:

Git intern fügt die Erstellung eines 'tracking branch' einen Abschnitt ähnlich dem folgenden in die Datei '<repository>/.git/config' ein:

```
[branch "3.8"]
	remote = origin
	merge = refs/heads/3.8
	rebase = true
```

```
git config branch.master.rebase true
```

Weitere Informationen in [http://stackoverflow.com/questions/11955305/what-is-the-meaning-of-pull-strategy-when-creating-a-branch-with-egit].

cherry-pick vs merge

Wenn *alle* Änderungen vom einem Branch in einen Anderen (z.B. den master) übertragen werden müssen, kann man das am besten mittels '`git merge`' machen. Das verhindert, dass man ständig daran denken muss, Änderungen auf diversen Branches einzuchecken, und sich zudem sicher sein kann, dass man am Ende keine Änderung verpasst /vergißt.

Der '`git cherry-pick`' wird in einem anderen Szenario verwendet: Dies ist dann Mittel der Wahl, wenn man *einzelne* Änderungen auf einem anderen Branch braucht. Das ist normalerweise der Fall, wenn man einen Bugfix von master dringend noch (auch) in einem *zu ende entwickelten* Branch braucht.