= GitKompakt

Diese stark verkürzte Einführung richtet sich in erster Linie an Nichttechniker, die in Projekten mitarbeiten wollen oder müssen welche Git zur Versionsverwaltung einsetzen.

== Vorwort

Mit

```
$ blabla
```

ist im folgenden immer ein Befehl `blabla` im Terminal bzw. einer Shell gemeint. Natürlich gibt's auch z.T. ganz hübsche GUIs (u.a. für Mac OS X) - aber da git selbst ein Kommandozeilenprogramm ist und die GUIs meist direkt darauf aufbauen und auch für verschiedene Rechnerplattformen ganz unterschiedlich sind, geht's auf dieser Wikiseite hier in erster Linie um das was immer gleich geht: die Kommandozeile.

Für die meisten Befehle (außer `clone`) muß man sich innerhalb eines von git verwalteten Verzeichnisses (auch „Arbeitskopie“ genannt, siehe unten) befinden (`cd meinprojekt` oder noch tiefer drin) - **NICHT** innerhalb des `.git` Verzeichnisses. Andernfalls bekommt man von den git Kommandos eine entsprechende Fehlermeldung. Kaputtgehen kann dabei nichts.

Eine ähnliche Beschreibung gibt's auch auf englisch auf der Git Website im Kapitel "Using git for collaboration" im Git Tutorial.

Die hervorragende Dokumentation von http://git.or.cz/#documentation wird damit keineswegs überflüssig - lediglich die Einsteigshürde für Nichttechniker sollte hierdurch entschärft werden. Git kann weit mehr als im Folgenden behandelt wird. Als Einstieg zu den höheren Weihen empfehle ich das Git Tutorial.

==Hilfe!

```
$ git help
```

==Wer bin ich? Damit git weiß mit wem es zu tun hat, sollte man sich gleich nach der Installation einmalig vorstellen:

```
$ git config --global user.name "Your Name Comes Here"
$ git config --global user.email you@yourdomain.example.com
```

==Erstmalig ein fremdes Archiv herunterladen

```
$ git clone <adresse>
```

Wobei die Adresse z.B. http://www.meinedomain.de/irgendwas/meinprojekt.git lautet. Auch andere als http:// Adressen sind möglich, Details dazu gibt's im Kapitel "GIT URLS" der git clone Doku.

Mit .git enden die Adressen traditionellerweise, technisch gesehen könnte da auch was anderes

stehen.

Ergebnis ist ein neues Verzeichnis mit dem Namen `meinprojekt` (ohne .git), das sowohl den aktuellen Stand des Projekts (die sogenannte Arbeitskopie) als auch (in einem Unterverzeichnis `.git`) eine komplette Kopie des gesamten Archivs incl. der vollständigen Versionshistorie in gepackter Form enthält. Das Archiv benötigt erstaunlich wenig Speicherplatz, da sämtliche Veränderungen in extrem platzsparender Weise gespeichert werden.

==Änderungen aus einem fremden Archiv herunterladen Falls das Archiv, das man per `git clone . . .` mal geholt hatte inzwischen Neues zu bieten hat, empfiehlt per

```
$ git pull
```

diese Neuigkeiten in's eigene Archiv herunterzuladen.

git pull Man Page

==Versionen im Archiv angucken

```
$ gitk --all
```

Möglicherweise ist auch ein auf git aufbauendes Werkzeug eine Hilfe. Im Folgenden geht's aber nur um das „originale“ git - die Kommandozeilenversion.

gitk Man Page

==Änderungen in's eigene Archiv sichern Vorher gucken was man selbst alles geändert hat:

```
$ git status
```

git status Man Page

Und ab in's Archiv damit:

```
$ git commit -a -m "Meine Kurzbeschreibung warum die Version jetzt besser
ist als zuvor oder warum ich sie sichern will."
```

git commit Man Page

Vorhandene Entwicklunglinien (Zweige) auflisten

```
$ git branch
```

wobei der aktuell aktive Zweig farblich hervorgehoben und mit einem * markiert ist.

git branch Man Page

==Eine neue Entwicklunglinie (Zweig) beginnen

```
$ git branch dasneuethema
```

legt einen neuen Zweig namens `dasneuethema` an, der auf dem zuvor aktiven Zweig aufbaut. Um wirklich in diesem neuen Zweig zu arbeiten, muß man noch:

==Von einem Zweig zum anderen Umschalten

```
$ git checkout dasneuethema
```

schaltet zu einem Zweig `dasneuethema` um. Wieder zum Hauptzweig kommt man mit

```
$ git checkout master
```

Der Zweig „master“ wird ist von Haus aus angelegt und bedeutet die „Hauptentwicklung“, bzw. den produktiven, offiziellen Stand des Projekts.

git checkout Man Page

==Änderungen aus einem Zweig in einen anderen (oder die Hauptrichtung) übernehmen Diese Aktion benötigt zwei git Aufrufe:

```
$ git checkout zweig_wo_solls_hin
```

Wenn's in die Hauptentwicklung einfließen soll, steht statt `zweig_wo_solls_hin` der Zweig `master`.

Und danach:

```
$ git merge zweig_wo_kommts_her
```

git merge Man Page

==Änderungen in's Projekt publizieren Dazu gibt't verschiedene Möglichkeiten, z.B.:

===Komplettstand per Email Der Holzhammer ist natürlich die geänderten Dateien per Mail an ein Projektmitglied, das Schreibzugriff auf das Archiv hat welches man sich per `git clone` geholt hatte zu schicken.

===Änderungen per Email Wesentlich sinnvoller ist, nur die Änderungen per Mail zu verschicken.

Siehe http://www.kernel.org/pub/software/scm/git/docs/git-format-patch.html

===Das eigene Archiv veröffentlichen Diese Möglichkeit dürfte wahrscheinlich am bequemsten sein, da man sich selbst nicht darum kümmern braucht wie die eigene Arbeit genau wieder in's Projekt einfließt.

Dazu genügt es zuerst

```
$ git update-server-info
```

aufzurufen und danach den **Inhalt** des **`.git`** Ordners z.B. per FTP auf einen Webserver in z.B. ein Verzeichnis `meinprojekt.git` hochzuladen und einem Projektmitglied die zugehörige http://... Webadresse zu nennen.

git-update-server-info Man Page

Howto, SCM, Git, Versionsverwaltung