Inhaltsverzeichnis

Git Tricks

Aktuellen Branch im Command Prompt anzeigen

passenden Code-Schnipsel als „Raw“ von Github downloaden und abspeichern

an die `.bashrc` die folgende Zeile anhängen

```
source ~/bin/git-prompt.sh
```

und die Zeilen zum Setzen des Prompts abändern:

```
if [ "$color_prompt" = yes ]; then
#PS1='${debian_chroot:+($debian_chroot)}\[\033[01;32m\]\u@\h\[\033[00m\]:\[\033[01;34m\]\w\[\033[00m\]\$ '
PS1='${debian_chroot:+($debian_chroot)}\[\033[01;32m\]\u@\h\[\033[00m\]:\[\033[01;34m\]\w\[\033[00m\]$(__git_ps1 " (%s)")]\$ '
else
    #PS1='${debian_chroot:+($debian_chroot)}\u@\h:\w\$ '
    PS1='${debian_chroot:+($debian_chroot)}\u@\h:\w$(__git_ps1 " (%s)")\$ '
fi
```

oder noch einfacher (https://coderwall.com/p/fasnya/add-git-branch-name-to-bash-prompt)

Add following lines to your ~/.bash_profile

```
parse_git_branch() {
     git branch 2> /dev/null | sed -e '/^[^*]/d' -e 's/* \(.*\)/ (\1)/'
}
export PS1="\u@\h \[\033[32m\]\w\[\033[33m\]\$(parse_git_branch)\[\033[00m\] $ "
```

Lokale Änderungen beim Pull überschreiben

Dieser Befehl überschreibt alle lokalen Modifikationen mit dem Stand vom Repository

```
git pull -s recursive -X theirs
```

Lokale Änderungen beim Pull behalten

Diese Sequenz dagegen zieht erst den Stand vom Server, aber um ihn dann mit den lokalen Änderungen zu überschreiben

```
# fetch from the default remote, origin
```

```
git fetch
# create a branch at your current master
git branch old-master
# reset to origin's master
git reset --hard origin/master
# merge your old master, keeping "our" (origin/master's) content
git merge -s ours old-master
```

Lokale Änderungen komplett entfernen

Die Standard- Variante:

```
# fetch from the default remote, origin
git fetch
# reset your current branch (master) to origin's master
git reset --hard origin/master
```

"Dry Run" eines Merge

Wie man weiß, kann ein Merge so richtig in die Hose gehen, wenn Files unterschiedlich sind, mit denen man gar nicht gerechnet hat. Nun hat git von Haus aus keinen „dry-run merge“ Befehl, aber mit etwas Kommandozeilenakrobatik bekommt man einen Diff auf die Unterschiede, die man dann per grep abklopfen kann, ob es einen Merge- Knall gibt oder das Ganze kollosionsfrei über die Bühne geht.

Im Einzelnen:

- `git fetch origin master` holt erstmal den letzten Stand auf die lokale Platte
- `git merge-base FETCH_HEAD master` liefert den commit-Hash, wo beide Branches getrennt wurden
- `git merge-tree mergebase master FETCH_HEAD` (mergebase ist die ID aus dem vorherigen Schritt)
- ein `| grep -A3 „changed in both“` holt dann die Annmerkungen raus, wo Files kollidieren

Als Einzeiler wird daraus dann ein `git merge-tree `git merge-base FETCH_HEAD development` FETCH_HEAD development | grep -A3 „changed in both“`

Merge "Ours" und "Theirs"

Wenn man schon gepullt hat und Merge Errors bekommt, kann man mit folgendem Befehlen entscheiden, ob man die eigene („ours“) oder die Version der anderen („theirs“) behalten möchte:

Die eigene Version

```
git checkout --ours -- <file>
```

Die version der Anderen:

```
git checkout --theirs -- <file>
```

Danach das `git add ..` nicht vergessen, um den Merge abzuschliessen

(Mehr Details auf https://stackoverflow.com/questions/914939/simple-tool-to-accept-theirs-or-accept-mine-on-a-whole-file-using-git)

Lokaler Commit Vorsprung

Zeigt den Unterschied der lokalen aktuellen Commits und dem Remote Server:

```
git log @{push}..
```

Wiederherstellen (Restore) früher gelöschter Dateien

(https://stackoverflow.com/a/30875442)

Find the last commit that affected the given path. As the file isn't in the HEAD commit, this commit must have deleted it.

```
git rev-list -n 1 HEAD -- <file_path>
```

Then checkout the version at the commit before, using the caret (^) symbol:

```
git checkout <deleting_commit>^ -- <file_path>
```

Dateien lokal behalten, aber im Repostory entfernen

see https://stackoverflow.com/a/3469805

```
git rm --cached -r somedir
```

Will stage the deletion of the directory, but doesn't touch anything on disk. This works also for a file, like:

```
git rm --cached somefile.ext
```

Afterwards you may want to add somedir/ or somefile.ext to your .gitignore file so that git doesn't try to add it back.

Git Pull with Submodule

- `git submodule update --init --recursive`. for the first time. All submodules will be pulled down locally. To update submodules, we can use:
- `git submodule update --recursive --remote`, or simply:
- `git pull --recurse-submodules`. The first one works for git version 1.8. 2 or above while the second one works for git version 1.7.
- `git submodule foreach --recursive git pull origin master` scheint am besten zu funktionieren...

Git Passwort permanent speichern

(für Windows)

```
$ git config --global credential.helper manager
$ git push http://example.com/repo.git
Username: <type your username>
Password: <type your password>

[several days later]
$ git push http://example.com/repo.git
[your credentials are used automatically]
```

für Linux (wird als Plaintext in `~/.git-credentials` gespeichert)

```
git config --global credential.helper store
```

Lokalen Status zeigen

checkGitStatus.sh

```
#!/bin/sh

UPSTREAM=${1:-'@{u}'}
LOCAL=$(git rev-parse @)
REMOTE=$(git rev-parse "$UPSTREAM")
BASE=$(git merge-base @ "$UPSTREAM")

if [ $LOCAL = $REMOTE ]; then
    echo "Up-to-date"
elif [ $LOCAL = $BASE ]; then
    echo "Need to pull"
elif [ $REMOTE = $BASE ]; then
    echo "Need to push"
else
    echo "Diverged"
```

```
fi
```

Git LFS alle File Extensions filtern und tracken

Dieser Einzeiler[1)] erzeugt eine Liste aller verwendeten File- Extension zum Tracken im fertigen `.gitattribute` Format

```
find . -type f | perl -ne 'print $1 if m/\.([^.\/]+)$/' | sort -u | awk '{print $1" filter=lfs diff=lfs merge=lfs -text"}' | sed 's/^/*./'
```

HTTPS self signed certificate Error unter Windows

Das Problem entsteht immer dann, wenn sich die hauseigene IT in den HTTPS- Datenverkehr gemogelt hat. Man kann dies beheben, indem man git die Benutzung der in Windows hinterlegten Firmen- internen Zertifikate erlaubt:

```
git config --global http.sslbackend schannel
```

Disable SSL verification in git repositories with self-signed certificate

Prepend GIT_SSL_NO_VERIFY=true before every git command run to skip SSL verification. This is particularly useful if you haven't checked out the repository yet.

einen zweiten Remote einrichten

den alten evtl. umbenennen

```
git remote rename origin backup
git remote -v
```

den neuen hinzufügen

```
git remote add origin <newURL>
git push --set-upstream origin main
```

uncommittete Directories suchen

Dieses kleine (Git-)Bash- Script sucht nach Unterverzeichnissen mit uncomitteten Änderungen

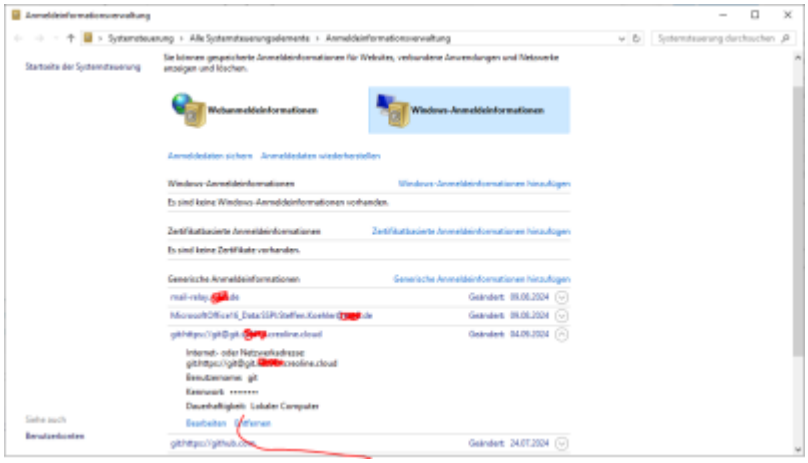
show_uncommited.sh

```
for i in $(find * -maxdepth 0 -type d);
do
    echo "====   $i ======="
    #git -C "$i" status
    git -C "$i" ls-files -m  --exclude-from=.gitignore
    #echo $?
    echo
    echo
    echo
done
```

Gespeichertes Token (Passwort) löschen

Windows

Läuft z.B. ein Token ab, dann bleibt das Alte im Windows Credential Manager gespeichert und wird immer weiter verwendet, d.h. man kann sich nicht mehr anmelden. Das alte Passwort muß erst gelöscht werden, und zwar in der Anmeldeinformationsverwaltung (schönes deutsches Wort..)- bzw. man kann im „Bearbeiten“- Modus das neue Token auch gleich eintragen, geht auch.



Linux

In der Kommandozeile löscht ein

```
   echo 'url=https://git.example.com' | git credential reject
```

den Cache, und danach gibt man beim nächsten Push/Pull das Passwort wieder neu ein

1)

https://stackoverflow.com/a/65607756