GIT

GIT-Guide useful commands

https://owenou.com/ten-things-you-didnt-know-git-and-github-could-do

minimaler clone

```
git clone --depth 1 https://github.com/frank-w/BPI-R2-4.14
```

bestimmten branch

```
git clone --depth 1 --single-branch --branch 5.4-main
https://github.com/frank-w/BPI-R2-4.14 5.4-main
```

mit „–single-branch“ werden nur Informationen zum gewählten branch heruntergeladen

```
$ git branch -a
* 5.4-main
  remotes/origin/5.4-main
```

möchte man trotzdem alle branchinformationen haben nimmt man stattdessen „–no-single-branch“, dadurch ist aber der Download größer (depth x anzahl_branches)

hat man nur die Sparvariante und möchte später wieder auf die anderen branches zugreifen:

```
git remote set-branches origin '*'
git fetch --depth 1
```

Sparse checkout

https://stackoverflow.com/a/59515426

log

geänderte Dateien

um nur die geänderten Dateien aufzulisten:

```
git log --name-only --oneline
```

Commit nach Autor suchen

```
git log --author="Frank Wunderlich"
```

alternativ wer es commited hat (hilfreich für Patches die via „git am“ angewendet wurden)

```
git log --oneline --committer=frank-w
```

Commits nach Text suchen

in der Commit-Message

```
git log --grep="hnat"
```

im Diff (Code-Änderung, ab Version 1.7.4)

```
git log -G"hnat"
```

alternativ (commits finden, die den Suchstring hinzufügen/löschen)

```
git log -S"hnat"
```

mehrere Suchkriterien

```
git log --all-match --grep="hnat" --author=Frank
```

alle commits eines Branches

```
git log hnat --not main --abbrev-commit
```

zeigt Änderungen in hnat an,welche nicht in branch main sind

remote log

```
git fetch
#zeigt die log des Branches master auf remote origin an
git log origin/master
#remote ermitteln (falls nicht origin):
git remote
```

eigenes Format

```
git log --pretty=format:'%C(Yellow)%h %C(Green)%ad %C(reset)%s
```

```
%C(Cyan)(%an)' --date=short
```

inklusive patch

```
git log --oneline -p
```

merges

zuerst merge-commit ermitteln, z.B. so:

```
git log --merges
```

danach kann man mit diesen Commit-Hash ermitteln, welche commits dran hängen

```
git log --no-merges --oneline <hash>^..<hash>
```

Beispiel als alias:

```
logm = !git log --no-merges --oneline $1^..$1
```

will man nur die Datei-Änderungen des kompletten merges sehen, kann man das so:

```
git show -m <hash>
```

diff

```
git log BRANCH --abbrev-commit
git diff COMMIT1..COMMIT2 >changes.patch
git diff COMMIT^ COMMIT
#branch-übergreifend für einzelne Datei
git diff 4.14-main..4.9-main -- arch/arm/boot/dts/mt7623.dtsi

#Verzeichnis ausschließen:
git diff
b543a94c2247bbf9e66ad9996f7d0273458faacd..bbbb5eb11e10e9fa4e0e3a620f8aa47e1e
8355c4 -- . ':(exclude)drivers/misc/mediatek/*' >wifi.diff

#mit remote vergleichen
branch=$(git rev-parse --abbrev-ref HEAD)
git diff $branch origin/$branch
```

show

```
git show -M HEAD^^
```

zeigt auch umbenannte Dateien an (-M = −−find-renames)

alternativ das in der git-config:

```
[diff]
   renames = copies
   renamelimit = 0
```

merge-commits

```
git show -m 34183ddd13db
```

zeigt alle Änderungen durch den merge, nicht nur die Änderungen, die aus Konflikten entstanden sind

bestimmte Version einer Datei

```
git show <treeish>:<file>
```

man kann die Ausgabe auch an einen externen Editor (hier gedit) senden, jedoch muss man das Syntax-highlighting neu einstellen (da ja keine Datei übergeben wird an deren Endung der Dateityp erkannt wird)

```
git show <treeish>:<file> | gedit -
```

alternativ kann man auch die Ausgabe in eine Temporäre Datei leiten und diese öffnen

```
file=<file>
git show <treeish>:path/to/$file > /tmp/$file
gedit /tmp/$file
```

auch committer-info zeigen

```
git show --format=fuller HEAD
```

commit

Änderungen anzeigen

```
git status
```

unversionierte Dateien hinzufügen

```
git add <Datei/Verzeichnis>
```

alles in Git übernehmen (-s=signieren)

```
git commit -as
```

nur bestimmte Dateien/Verzeichnisse

```
git commit <Datei/Verzeichnis>
```

Nur bestimmte Teile

```
#Datei/Verzeichnis optional
git add -p <Datei/Verzeichnis>
git commit
```

letzten commit überarbeiten (sollte noch nicht gepusht sein)

```
git add <Datei>
git commit --amend
git commit --amend --author="Name <name@mail.com>" #Autor setzen
```

ältere commits ändern (noch nicht gepushed):

```
git rebase -i HEAD~X
# X ist ie Anzahl der Commits die man zurückgehen möchte alternativ <sha1>^
um bis zu einem bestimmten commit zurückzugehen
# git rebase -i --root #wenn der erste commit geändert werden soll
# in die entsprechende commit-zeile gehen und nur pick in reword ändern,
# wenn edit statt reword:
git commit --amend
# rebase abschließen:
git rebase --continue
```

Quelle

commit in welchen tags/branches

```
git branch --contains  528222d853f9
git tag --contains  528222d853f9
```

Datei aus staging entfernen

wenn versehentlich Datei/Verzeichnis via „git add –all“ hinzugefügt wurde (noch nicht committed)

```
git reset HEAD -- Datei
```

Datei aus vorherigen Commit entfernen

wenn schon committed (und noch nicht gepushed) wurde

```
git reset HEAD^ .gitignore
git commit --amend
```

tagging

```
git tag -a v4.14.10-1 -m "first stable kernel"
```

```
git tag v4.16-rc1
git push origin v4.16-rc1
```

```
git tag -l
```

lokale Änderungen ignorieren

soll eine Datei im remote-repo nicht mehr auf Änderungen überwacht werden:

```
git update-index --assume-unchanged [<file> ...]
```

tracking wieder aktivieren:

```
git update-index --no-assume-unchanged [<file> ...]
```

push

den aktuellen Branch auf das/die remote-Repo

```
git push
```

alle Branches:

```
git push --all origin
```

erzwungen (um commits entfernt zu löschen/überschreiben)

```
#git push --force
git push --force-with-lease #überschreibt fremde commits nicht einfach
```

https://www.whatsdoom.com/posts/2016/09/07/safely-force-pushing-with-git/

alternativ schauen, was sich verändert hat und rebase des lokalen trees:

```
git fetch #remote aktualisieren, falls noch nicht gemacht
git log 5.4-main..origin/5.4-main #unterschiede anzeigen (commits, die
remote vorhanden sind, aber nicht lokal)
git rebase origin/5.4-main
```

lokales Repo aktualisieren

```
git fetch
git merge --ff-only origin/master
```

(funktioniert nur, wenn lokal nichts geändert wurde)

oder

```
git rebase --preserve-merges v4.20-rc7
#neueres git (Ubuntu 20.4)
git rebase --rebase-merges v5.7
```

(holt die Änderungen vom remote und rebased die eigenen „on-top“)

alternative (aktuellen branch im losgelösten Head aktualisieren):

```
git rebase --onto v5.3-rc1 <erster commit>^ <letzter commit>
git checkout -b <neuer branch>
```

oder (remote mit lokal mergen inkl. merge-commit)

```
git pull
```

oder (lokalen branch mit dem remote überschreiben)

```
git reset --hard origin/branch
```

merge

```
git merge --ff-only new-feature
```

https://shinglyu.github.io/web/2018/03/25/merge-pull-requests-without-merge-commits.html

möchte man immer einen merge-commit, auch wenn dieser per default nicht angelegt wird:

```
git merge --no-ff new-feature
```

den lokalen branch auf den stand des remote updaten (wenn lokal teilmenge des remote-branches)

```
git merge --ff-only @{upstream}
```

merge-commit finden

https://stackoverflow.com/a/30998048

revert

Merge rückgängig machen

```
git revert -m 1 <merge commit hash>
```

rebase

https://www.ralfebert.de/git/rebase/

```
git rebase -i <commit>
```

History nach <commit> neu schreiben

```
git rebase -i <commit>^
```

schließt commit mit ein

wurde vor dem Rebase bereits gepusht wird ein erneuter push abgewiesen (rejected), statt einem force sollte man hier folgendes verwenden, da damit zusätzliche commits (anderer Benutzer) berücksichtigt werden

```
git push --force-with-lease
```

rebase als merge

```
git checkout fix_types
git rebase master
git checkout master
git merge fix_typos --ff-only
```

https://coderefinery.org/blog/2020/04/24/rebase-vs-merge/

als git alias (~/.gitconfig):

```
rebasefeature = "!f() { CUR=$(git rev-parse --abbrev-ref HEAD);FEAT=$1;echo
$CUR $FEAT; git checkout $FEAT; git rebase $CUR; git checkout $CUR; git
```

```
merge $FEAT --ff-only; }; f"
```

rebase rückgängig machen

```
git reflog
git reset --hard HEAD@{5}
```

https://stackoverflow.com/a/135614

datei entfernen

```
git reset HEAD^ path/to/unwanted_file
git commit --amend
```

oder:

```
git reset --soft HEAD^
git reset HEAD path/to/unwanted_file
git commit -c ORIG_HEAD
```

wenn datei in späteren Commits verwendet wird, muss nun die Datei wieder separat hinzugefügt werden, sonst schlägt der rebase fehl

```
git co 85468fa659ac .config
git commit
```

der SHA1 ist der ursprüngliche, wo die Datei hinzugefügt wurde, dieser steht beim rebase (Stopped at 85468fa659ac...)

commit splitten

im Rebase den jeweiligen commit mit e kennzeichnen statt pick

wenn bei dem commit angekommen:

```
git reset HEAD^
```

um alle Datei-Änderungen dieses Commits zurückzusetzen

sollen komplette Datei-Änderungen übernommen werden, einfach via „git commit <datei>“ („-c ORIG_HEAD“ übernimmt die Beschreibung des ursprünglichen zu splittenden Commit)

sollen nur bestimmte Patch-Bereiche übernommen werden mit „git add -p“ arbeiten +commit

Basis aktualisieren

Angenommen,ein Feature-branch basiert auf master und am master ändert sich etwas. Manchmal ist es sinnvoll oder nötig,den master-teil des Feature-branches auf den aktuellen Stand zu bringen.

```
git rebase master
```

Im feature-branch durchführen. Evtl. Vorhandene Konflikte auflösen (editieren,git add datei,git rebase –continue)

aufräumen

```
git reflog expire --expire=now --all
git gc --aggressive --prune=now
```

falls das mit Fehler abbricht (out of memory):

```
fatal: Out of memory, realloc failed
error: failed to run repack
```

https://stackoverflow.com/questions/4826639/repack-of-git-repository-fails

- neues ServerRepo anlegen (git init –bare)
- git-config erweitern

```
[http]
        receivepack = true
[pack]
        windowMemory = 16m
        packSizeLimit = 56m
        deltaCacheSize = 1
        threads = 1
```

- ggf. „git repack -a -d“, dafür habe ich ein alias definiert

  ```
  cleanup = "!f() { git reflog expire --expire=now --all; git gc --
  aggressive --prune=now;git repack -a -d; }; f"
  ```

- via scp die neu gepackten (auf kleine chunks) auf den Server kopieren

```
scp .git/objects/pack/* $server:/$pfad/$repo/objects/pack/
scp -r .git/packed-refs $server:/$pfad/$repo/
```

- Rechte auf dem Server anpassen (chmod -R user:group .;chmod -R g+w .)
- jetzt erst den push durchführen (ggf. mit ssh)

```
git push --mirror ssh://$server/$pfad/$repo
```

```
git config --global core.packedGitLimit 128m
```

```
git config --global core.packedGitWindowSize 128
```

in der gitconfig sieht das dann so aus:

```
[core]
  packedGitLimit = 128m
  packedGitWindowSize = 128
```

nachdem ich irgendwann weiterhin out-of-memory Fehler bekommen hatte inklusive oom-reaper (Prozess gekillt wegen zu wenig Speicher) konnte ich das Problem mit einem zusätzlichen SWAP-Speicher (auf ssd) lösen

```
swapfile=/var/swap.img
if [[ ! -e $swapfile ]];then
  dd if=/dev/zero of=$swapfile bs=1M count=4096
fi
chmod 0600 $swapfile
mkswap $swapfile
swapon $swapfile
```

Quelle: https://stackoverflow.com/a/54755601

filter-branch

```
git filter-branch --tree-filter 'rm filename' HEAD
git filter-branch -f --tree-filter "rm -rf external/bin/lastrunAlarms" --
prune-empty HEAD

git filter-branch --tree-filter "test -f \"$filename\" && sed -i
\"s/$filter//g\" \"$filename\" || echo \"skipping $filename\"" -- --all
```

https://stackoverflow.com/questions/7194939/git-change-one-line-in-file-for-the-complete-history

whitespace-fix

```
#make sure no patch-files are in your current directory
git format-patch sha1^
git checkout sha1^
git checkout -b branch-ws
git am --whitespace=fix *.patch
```

debugging

bisect

* https://stackoverflow.com/questions/3179498/how-can-i-cut-down-a-git-bisect-run-using-file-paths

Branch

branch erzeugen

```
git checkout -b <branch>
git push --set-upstream origin <branch>
```

branch löschen

entfernter Branch:

```
git push origin --delete <branch>
```

lokal:

```
git branch -d <branch>
```

evtl. in Schleife bei versehentlichem upload (Datei mit 1 branch pro zeile)

```
while read l; do git push gitlab --delete $l; done < new_branch_list.txt
```

upstream-Verlinkung löschen

```
git branch --unset-upstream <LOCALBRANCH>
```

branches auflisten

```
git branch -vv
```

zeigt alle branches (lokal/remote) mit letztem commit und push-info (x commits hinterher/voraus)

Branches mit bestimmten commit finden

```
git branch --contains <commit-id>
```

remote branches aktualisieren

```
git remote update origin --prune
```

löscht alle Referenzen auf remote-branches, welche nicht mehr existieren

remote branch sichern

bevor ein entfernter branch überschrieben / gelöscht wird, kann man ihn so in einen anderen branch (lokal) sichern

```
git checkout -b 5.8-rc-old origin/5.8-rc
git branch --unset-upstream
```

Datei/Verzeichnis aus anderem Branch in aktuellen

```
git checkout otherbranch myfile.txt
```

commit aus anderen Branch

```
git cherry-pick sha_hash
```

um eine Art Squash-merge zu machen (2+ commits in einem zusammenfassen)

```
git cherry-pick -n erster_commit
git cherry-pick -n zweiter_commit
git commit -a
```

Konflikt-Markierung

https://stackoverflow.com/questions/7901864/git-conflict-markers

```
<<<<<<< HEAD:file.txt
Hello world #local
=======
Goodbye #remote
>>>>>>> 77976da35a11db4580b80ae27e8d65caf5208086:file.txt
```

pull-request in neuen Branch

Pullrequest #13 in neuen Branch unsquashed

```
git fetch origin pull/13/head:unsquashed
```

```
Von https://github.com/frank-w/BPI-R2-4.14
 * [neue Referenz]   refs/pull/13/head -> unsquashed
```

Branch umbenennen

```
OLDBR=old-name
NEWBR=new-name
# 1. Rename your local branch.
# If you are on the branch you want to rename:
git branch -m $NEWBR
# If you are on a different branch:
git branch -m $OLDBR $NEWBR
#2. Delete the old-name remote branch and push the new-name local branch.
git push origin :$OLDBR $NEWBR
#3. Reset the upstream branch for the new-name local branch.
# Switch to the branch and then:
git push origin -u $NEWBR
```

Quelle: https://multiplestates.wordpress.com/2015/02/05/rename-a-local-and-remote-branch-in-git/

Branch wechseln

```
git checkout branch
```

stash

https://git-scm.com/docs/git-stash/

„git stash“ legt Änderungen zur Seite, um trotzdem den Branch wechseln zu können

neben der quick&dirty-variante kann man auch via

```
git stash push -m "Beschreibung"
```

einen Name vergeben

```
#vorhandene Stashes anzeigen
git stash list
stash@{0}: WIP on gmac: e4f9258 added missing mt7623a-rfb-emmc.dts
stash@{1}: WIP on main: f5b5b75 decreasing debug-level of Wifi-driver

#Änderungen in einem Stash anzeigen
git stash show -p stash@{1}
diff --git a/.gitignore b/.gitignore
index d31ce3fa..19ca85c 100644
--- a/.gitignore
```

```
+++ b/.gitignore
@@ -42,7 +42,7 @@ Module.symvers
 arch/arm/boot/zImage-dtb
 uImage
 *.rej
-
+u-boot/

#stash löschen
git stash drop stash@{1}
Gelöscht stash@{1} (15af1209b06ff52e3b4ed6cf898d0e394b2da67c)

#übernehmen des letzten Stashes in die Arbeitskopie
git stash apply
#apply + drop
git stash pop

#neuen branch (name=stashbranch) aus dem stash (originalposition) anlegen
git stash branch stashbranch stash@{0}
```

Mbox-Dateien

häufig liegen Patches/-Serien als Mbox-Datei (Email-Format) vor und diese lassen sich recht komfortabel mit git am importieren. Dabei werden Author und die Commit-Nachricht auch übernommen. Bei Serien wird für jede Mail ein einzelner Commit angelegt.

```
git am mbox.patch
#whitespace-fix
git am --whitespace=fix *.patch
```

interessant wird es, wenn ein Patch fehlschlägt. normalerweise bricht git dann ab und man muss den gesamten Patch der jeweiligen Mail manuell importieren. Ich behelfe mir mit der folgenden Vorgehensweise:

als erstes das Arbeitsverzeichnis sauber machen (git Status sollte keine Dateien/Verzeichnisse anzeigen)...am besten das Patchen in einem eigenen Branch durchführen.

```
#reject-parameter führt den Patch soweit wie möglich aus und man muss nur
den Chunk manuell einpflegen, der fehlgeschlagen ist
git am --reject ~/Downloads/ubootv3_Add-U-Boot-support-for-MediaTek-SoCs---
MT7623n-MT7629.patch
```

die Ausgabe sieht etwa so aus:

```
Patch include/dt-bindings/power/mt7623-power.h sauber angewendet
Wende an: arm: MediaTek: add basic support for MT7629 boards
Prüfe Patch arch/arm/Kconfig ...
error: bei der Suche nach:
          targeted at media players and tablet computers. We currently
```

```
          support the S905 (GXBaby) 64-bit SoC.

config ARCH_LPC32XX
        bool "NXP LPC32xx platform"
        select CPU_ARM926EJS

error: Anwendung des Patches fehlgeschlagen: arch/arm/Kconfig:658
Patch-Bereich #2 erfolgreich angewendet bei 1423 (-33 Zeilen versetzt)
```

hier sieht man, dass ein Patch in arch/arm/Kconfig fehlgeschlagen ist. Dabei wird eine rej-Datei mit gleichem Namen angelegt (wie beim normalen Patch, nur git sagt es nicht). In dieser Datei ist jetzt nur der Abschnitt (Chunk) der nicht eingebracht werden kann. Dieser muss jetzt manuell eingefügt werden und danach sollte rej-Datei gelöscht werden. Das wiederholen für jede Datei die nicht komplett gepatcht werden konnte.

sind alle Patches angewendet, müssen die Änderungen noch übernommen werden (in das staging) via „git add <verzeichnis/datei>“

zum Schluss wird „git am –continue“ aufgerufen um den aktuellen Commit abzuschließen und ggf. mit dem nächsten anzufangen

remote repos

am Beispiel, wie ich einen Branch für eine neue Kernel-Version anlege

```
git checkout 4.18-main
git reset --hard v4.18 #back to last mainline
git checkout -b 4.19-rc #create new branch for new kernel
git remote add torvalds
https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git
git fetch torvalds
git merge v4.19-rc1
git push --set-upstream origin 4.19-rc
#add additional files
git checkout 4.18-main build.sh
git checkout 4.18-main arch/arm/configs/mt7623n_evb_fwu_defconfig
git add build.sh arch/arm/configs/mt7623n_evb_fwu_defconfig
git commit
git push
```

fremdes repo einbinden und branches nutzen

```
git remote add vdorst https://github.com/vDorst/linux-1.git
git fetch --all
git log --oneline vdorst/5.0-phy
git checkout vdorst/5.0-phy #losgelöster head, da branch lokal noch nicht
existiert
git checkout -b 5.0-phy #branch lokal anlegen
```

```
git push --set-upstream origin 5.0-phy #ins eigene remote-repo pushen
```

mehrere Push-Repos

```
git remote set-url origin --push --add http://www.qnap.loc:7080/git/web_brk
git remote set-url origin --push --add http://www.pi.loc/git/web_brk
```

die git-config sieht nun so aus:

```
[remote "origin"]
        url = http://www.qnap.loc:7080/git/web_brk
        fetch = +refs/heads/*:refs/remotes/origin/*
        pushurl = http://www.pi.loc/git/web_brk
        pushurl = http://www.qnap.loc:7080/git/web_brk
```

hier muss man jetzt aufpassen, da in das ursprüngliche Repo (hinter url) nicht mehr gepusht wird...dieses muss ebenfalls als push-url vorhanden sein. Es ist dabei unerheblich, ob die .git/config angepasst wurde oder git remote verwendet wurde.

```
$ git remote -v
origin  http://www.qnap.loc:7080/git/web_brk (fetch)
origin  http://www.pi.loc/git/web_brk (push)
origin  http://www.qnap.loc:7080/git/web_brk (push)
```

git format-patch

```
git format-patch -vX sha1_from^..sha1_to --cover-letter
```

in 0000-cover-letter.patch wird dann die Beschreibung der Patch-Serie reingeschrieben

```
git format-patch sha1_from^..sha1_to --subject-prefix="RESEND PATCH v2" --
to="email" --cc="other email"
```

git send-email

```
sudo apt-get install git-email
```

https://www.freedesktop.org/wiki/Software/PulseAudio/HowToUseGitSendEmail/

die History (changes since vX) unter die Signed-off-by mit — getrennt

patch erzeugen (Version X von sha1 bis sha2 mit Erzeugung des Patch#0):

```
git format-patch -vX --cover-letter sha1^..sha2
```

Patch prüfen:

```
scripts/checkpatch.pl 000x-test.patch
```

Empfänger ermitteln:

```
scripts/get_maintainer.pl --norolestats *.patch
scripts/get_maintainer.pl v6-00{01..13}*.patch|sort -u
```

hier die committer rausnehmen und in den coverletter eintragen (1 mailingliste als To und die anderen als CC)

```
To: linux-mediatek@lists.infradead.org
Cc: devicetree@vger.kernel.org,
    linux-arm-kernel@lists.infradead.org,
    ...
```

und senden

```
git send-email --to-cover --cc-cover v6-*.patch
```

http://nickdesaulniers.github.io/blog/2017/05/16/submitting-your-first-patch-to-the-linux-kernel-and-responding-to-feedback/

falls man jemanden vergessen hat, kann man die serie nochmal so schicken (vorher im coverletter die Mailadressen entfernen und ggf. auf das Nachsenden hinweisen):

```
git send-email --to=linux-arm-kernel@lists.infradead.org --suppress-cc=all
v6-*.patch
```

Config

```
git config --list --show-origin
```

Benutzer/Email konfigurieren

```
git config --global user.name "Frank Wunderlich"
git config --global user.email @gmx.de
```

Editor ändern

falls für git nicht der standard-Editor verwendet werden soll

```
git config --global core.editor nano
```

Benutzername speichern

in .git/config die Zeile url unter „remote origin“ anpassen

```
[remote "origin"]
    url = https://frank-w@github.com/frank-w/u-boot.git
    fetch = +refs/heads/*:refs/remotes/origin/*
```

Passwort-Cache

Anmeldedaten die nächsten 5 Minuten speichern

```
git config credential.helper 'cache --timeout=300'
```

windows

zurücksetzen von BN/PW:

```
rundll32.exe keymgr.dll, KRShowKeyMgr
```

ggf. manager neu setzen (falls openssh-dialog kommt)

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

Alias

```
  git config alias.co checkout
  git config --global alias.ren '!f() { echo \"$1\" \"$2\"; }; f'
```

Beispiel Branch umbenennen:

```
git config --global alias.ren '!f() { git branch -m $1 $2;git push origin
:$1 $2;git push origin -u $2; }; f'
```

alternativ:

```
  nano .git/config
```

```
[alias]
    st = status
    co = checkout
    kver = "!f() { make kernelversion; };f"
    ckver = "!f() { git commit -a -m \"update to $(make kernelversion)\";
};f"
    current = rev-parse --abbrev-ref HEAD
```

```
    remote-diff = !git diff $1@{upstream}..$1@{0}
    nextref = "!f() { git log --reverse --ancestry-path --pretty=%H $1..HEAD
| head -${2:-1} | tail -1; }; f"
    cleanup = "!f() { git reflog expire --expire=now --all; git gc --
aggressive --prune=now;git repack -a -d; }; f"
#    logone = !git log --oneline
    logone = log --pretty=format:\"%C(auto,yellow)%h %C(auto,blue)%ad
%C(auto,reset)%s\" --date=short
    logm = !git log --no-merges --oneline $1^..$1 #commits, welche zu einem
merge-commit gehören
    pushnew = "!git push origin $(git rev-parse --abbrev-ref HEAD);"
```

Farben

```
git config --global --add color.ui true
#show whitespace-errors in git diff
git config diff.wsErrorHighlight all
```

umbenannte Dateien

git show zeigt normalerweise umbenannte Dateien als add/delete, um per default die renames anzuzeigen (wie git show -M):

```
[diff]
   renames = copies
   renamelimit = 0
```

bash-Integration

in ~/.bashrc das vorhandene PS1 anpassen:

```
parseGitBranch() {
    git rev-parse --abbrev-ref HEAD 2> /dev/null | sed -e 's/\(.*\)/(\1)/'
}

if [ "$color_prompt" = yes ]; then
PS1='${debian_chroot:+($debian_chroot)}\[\033[01;32m\]\u@\h\[\033[00m\]:\[\0
33[01;34m\]\w\[\033[00m\] $(parseGitBranch)\n\$ '
else
    PS1='${debian_chroot:+($debian_chroot)}\u@\h:\w $(parseGitBranch)\n\$ '
fi
```

Github

https://help.github.com/articles/duplicating-a-repository/

GIT-Server

um pushen zu können muss auf dem „Server“ ein bare-repo sein. Hier gibt es kein .git-Verzeichnis und dessen Dateien liegen in der obersten Ebene. Das Bare-Repo kann einfach nur ein anderer Ordner sein oder via http/ssh freigegeben sein

Lighttpd

🔧 Fix Me!

aktuell gibt es bei größeren Repos (z.b. mein Kernel-repo) noch einen 400-error beim push...

```
error: RPC failed; HTTP 400 curl 22 The requested URL returned error: 400
Bad Request
fatal: The remote end hung up unexpectedly
```

am Server:

```
(connections-glue.c.403) chunked data size too large -> 400
```

das ist ein Bug in lighttpd, welcher zwar gefixt, aber noch nicht in debian stretch integriert ist

https://redmine.lighttpd.net/issues/2854 (Zielversion:1.4.49)

```
lighttpd -v
lighttpd/1.4.45 (ssl) - a light and fast webserver
Build-Date: Jan 14 2017 21:07:19
```

evtl. auch https://redmine.lighttpd.net/boards/3/topics/6884

mit dem Lighttpd-Packet aus Buster (Nachfolger von Stretch) bricht der Transfer nicht ab, da dieses aber die libc mitzieht und man sich da leicht das ganze System zerlegt habe ich die aktuelle 1.4.51 kompiliert und in das Stretchpacket reingebastelt

```
#die neue Version runterladen, kompilieren und als deb packen
#quellen in /etc/apt/sources.list nochmal als deb-src hinzufügen
(vorhandene Zeilen kopieren und das -src ergänzen)
```

```
apt-get update
apt-get build-dep lighttpd
apt-get install wget checkinstall
wget
https://download.lighttpd.net/lighttpd/releases-1.4.x/lighttpd-1.4.51.tar.g
z -P /usr/local/src/
cd /usr/local/src/
sha256sum lighttpd-1.4.51.tar.gz #optional
tar -xf lighttpd-1.4.51.tar.gz
cd lighttpd-1.4.51
./configure --prefix=/usr --with-openssl --with-openssl-libs=/usr/lib/arm-
linux-gnueabihf/  --with-webdav-props --with-webdav-locks
make

#lief das fehlerfrei durch per checkinstall ein Packet bauen, damit es
nicht lokal installiert werden muss
checkinstall
#die Frage nach der Dokumentation mit n beantworten (wenn schon bestätigt
wurde kann das Verzeichnis doc-pak gelöscht werden, dann wird man erneut
gefragt)

  Should I create a default set of package docs?  [y]: n

#beim ersten Aufruf sollte noch eine Beschreibung eingegeben werden...
#diese wird hier aber nicht benötigt, da später nur das usr-Verzeichnis
herauskopiert wird und
#der Rest (modifiziert) vom alten Packet verwendet wird.
#Ich habe einfach nur "lighttpd (1.4.51)" eingetragen und mit Strg+D
beendet

#alles auf default lassen, da damit nur make install ins Packet reinläuft

#wenn fertig dann die deb nach /usr/local/src kopieren und wieder in das
Verzeichnis zurückgehen
cp *.deb ../
cd ../

#originalpacket runterladen und entpacken
root@stretch-dev:/usr/local/src# wget
http://ftp.de.debian.org/debian/pool/main/l/lighttpd/lighttpd_1.4.45-1_armh
f.deb
#i386:http://ftp.de.debian.org/debian/pool/main/l/lighttpd/lighttpd_1.4.45-
1_i386.deb
#amd64:http://ftp.de.debian.org/debian/pool/main/l/lighttpd/lighttpd_1.4.45
-1_amd64.deb
root@stretch-dev:/usr/local/src# dpkg-deb -R lighttpd_1.4.45-1_armhf.deb
lighttpd_1.4.51_test
root@stretch-dev:/usr/local/src# ls -l lighttpd_1.4.51_test
insgesamt 20
drwxr-xr-x 2 root root 4096 Jan 14  2017 DEBIAN
drwxr-xr-x 6 root root 4096 Jan 14  2017 etc
```

```
drwxr-xr-x 3 root root 4096 Jan 14  2017 lib
drwxr-xr-x 5 root root 4096 Jan 14  2017 usr
drwxr-xr-x 5 root root 4096 Jan 14  2017 var
#dort nun die Versionsnummer in der DEBIAN/control ändern
#Version: 1.4.51-1
#da auch php5-cgi nicht mehr existiert habe ich das auch angepasst (auf
php-cgi)

#das neue Packet entpacken wir jetzt und kopieren den inhalt über das alte
dpkg-deb -R lighttpd_1.4.51-1_armhf.deb lighttpd_1.4.51_unpack
cp -r lighttpd_1.4.51_unpack/usr/* lighttpd_1.4.51_test/usr/
#zum Schluss wieder packen
dpkg-deb -b lighttpd_1.4.51_test lighttpd_1.4.51-2_armhf.deb
```

lighttpd_1.4.51-2_armhf.deb
lighttpd_1.4.51-2_amd64.deb

repo anlegen:

```
mkdir reponame
cd reponame
git init --bare
chown -R www-data:frank .
#chmod -R g+w .
```

in der config des server-repo nicht vergessen:

```
[http]
        receivepack = true
```

ggf. postreceive-hook für automatisches checkout:

post-receive.txt
(ohne .txt, ausführbar im Ordner hooks des server-repos)

/etc/lighttpd/conf-available/11-git.conf

```
#https://git-scm.com/docs/git-http-backend
server.modules += (
  "mod_cgi",
#  "mod_alias",
#  "mod_auth",
  "mod_setenv")

alias.url += ( "/git" => "/usr/lib/git-core/git-http-backend" )
$HTTP["url"] =~ "^/git" {
        cgi.assign = ("" => "")
        setenv.add-environment = (
                "GIT_PROJECT_ROOT" => "/mnt/vcs/git",
```

```
                "GIT_HTTP_EXPORT_ALL" => "1"
        )
}

#anonymous read+auth write
#$HTTP["querystring"] =~ "service=git-receive-pack" {
#       include "git-auth.conf"
#}
#$HTTP["url"] =~ "^/git/.*/git-receive-pack$" {
#       include "git-auth.conf"
#}

#auth read+write
$HTTP["url"] =~ "^/git/private" {
        include "git-auth.conf"
}
```

die git-auth.conf muss noch angelegt werden (hier ohne Funktion = unbeschränkter Zugriff):

/etc/lighttpd/git-auth.conf

```
#auth.require = (
#        "/" => (
#                "method" => "basic",
#                "realm" => "Git Access",
#                "require" => "valid-user"
#               )
#)

#auth backend definieren
```

```
lighty-enable-mod git
```

zum Schluss noch den Webservice neustarten

```
service lighttpd restart
```

nun kann man mit folgenden Befehl ein vorhandenes Repo auf den Server spiegeln:

```
git push --mirror http://server/git/kernel
```

ggf. danach bei dem Repo das neue remote-repo hinzufügen

sollte nach dem Upgrade von 1.4.45 auf eine höhere Version trotz gleicher Konfig das Repo nicht mehr gefunden werden ist vermutlich in der git-config für lighttpd „GIT_HTTP_EXPORT_ALL“ nicht auf „1“ sondern „“ (auch of zu finden), siehe https://stackoverflow.com/questions/52891023/lighttpd-git-http-backend-setenv-issue

bricht git mit folgender Meldung ab:

```
error: RPC failed; HTTP 411 curl 22 The requested URL returned error: 411
```

```
Length Required
```

sollte man den post-Puffer einstellen:

```
git config http.postBuffer 524288000
#oder global
git config --global http.postBuffer 524288000
```

automatischer Checkout bei Push

hooks/post-receive:

```
#!/bin/bash
TARGET="/var/www/html/master"
GIT_DIR="/var/www/html/repo.git"
BRANCH="master"

while read oldrev newrev ref
do
  # only checking out the master (or whatever branch you would like to
deploy)
  if [[ $ref = refs/heads/$BRANCH ]];
  then
    echo "Ref $ref received. Deploying ${BRANCH} branch to production..."
    git --work-tree=$TARGET --git-dir=$GIT_DIR checkout -f
  else
    echo "Ref $ref received. Doing nothing: only the ${BRANCH} branch may be
deployed on this server."
  fi
done
```

optional checkout filtern:

```
git config core.sparseCheckout true
```

info/sparse-checkout (alles außer Verzeichnis external)

```
/*
!external/
```

automatisches build

ich habe auf meinem NAS ein (sehr simples) deployment eingerichtet, welches mir nach einem Push automatisch mein Kernel-repo baut.

dazu muss ich natürlich erstmal die Abhängigkeiten installieren, die meine build.sh benötigt (siehe Readme.md). weiterhin benötigt ccache ein temp-Verzeichnis im home des Users unter dem die

build.sh läuft...da dies bei mir www-data ist (home=/var/www) musste ich das Verzeichnis hier anlegen und entsprechend die Rechte setzen.

Bei mir läuft der GIT-Server in einer VM, von daher bedenkenlos möglich ;) sonst sollte zumindest der Deploy-Prozess (und die installierten Tools) in eine VM gepackt werden.

der git-hook selbst sieht dann so aus (da ich nicht nur den Master-Branch bauen möchte sind die Code-Stellen auskommentiert):

```
#!/bin/bash
TARGET="/mnt/vcs/git_repo/BPI-R2-Linux-master"
GIT_DIR="/mnt/vcs/git_repo/BPI-R2-Linux"
BRANCH="master"

while read oldrev newrev ref
do
  # only checking out the master (or whatever branch you would like to
deploy)
#  if [[ $ref = refs/heads/$BRANCH ]];
#  then
     BRNCH=${ref//refs\/heads\//} #without "A" to avoid conflict with filter
    echo "Ref $ref ($oldrev => $newrev) received. Deploying
${BRNCH} branch to production..."
    mkdir -p $TARGET/$newrev
    git --work-tree=$TARGET/$newrev --git-dir=$GIT_DIR check
out -f $BRNCH
    echo "starting deploy..."
    (
      echo "deploy it..."
      cd $TARGET/$newrev
      ./build.sh importconfig
      ./build.sh build
      ./build.sh pack
      #...
    ) &> $TARGET/$newrev/deploy.log &
#  else
#    echo "Ref $ref received. Doing nothing: only the ${BRANCH} branch may
be deployed on this server."
#  fi
done
```

die Befehle zum kompilieren ließen sich natürlich in eine separate Datei auslagern, die im Git selbst liegt...auch die Bedingung, welche Branches gebaut werden sollen...ich wollte es aber einfach halten ;)

das wichtigste, was fehlt ist eine Warteschlange, falls der build-prozess noch läuft. hier kann man sich entweder über eine run-datei helfen oder man schaut, ob der langlaufende Prozess noch läuft

```
if [[ "$(pidof make)" != "" ]]; then echo "build running"; fi
if [[ "$(pidof tar)" != "" ]]; then echo "pack running"; fi
```

für beide Fälle müsste der build-Prozess geparkt werden

webbasierte Git-Tools

GITList

https://github.com/klaussilveira/gitlist/wiki/Screenshots

lighttpd

ich habe gitlist im document_root entpackt mit name glist (damit es nicht mit /git kollidiert)

in der /etc/lighttpd/lighttpd.conf muss dann folgendes hinzugefügt werden, damit es funktioniert

```
url.rewrite-once = (
    "^/glist/themes/.+" => "$0",
    "^/glist/favicon\.ico$" => "$0",
    "^/glist(/[^\?]*)(\?.*)?" => "/glist/index.php$1$2"
)
```

gefolgt von einem

```
service lighttpd restart
```

default-branch setzen

```
git symbolic-ref HEAD refs/heads/branch
```

Beschreibung

die Datei „description“ in jedem Repo-Ordner bearbeiten

Gitweb

https://git-scm.com/book/de/v1/Git-auf-dem-Server-GitWeb

/etc/gitweb.conf

```
$projectroot = "/mnt/vcs/git_repo/";
$project_maxdepth = 1;
```

/etc/lighttpd/conf-available/12-gitweb.conf

```
server.modules += ( "mod_alias", "mod_cgi", "mod_redirect", "mod_setenv" )
#url.redirect += ( "^/webgit$" => "/webgit/" )
alias.url += ( "/webgit/" => "/usr/share/gitweb/" )
$HTTP["url"] =~ "^/webgit/" {
       setenv.add-environment = (
               "GITWEB_CONFIG" => "/etc/gitweb.conf",
               "PATH" => env.PATH
       )

       server.indexfiles = ( "gitweb.cgi" )
}
```

git-php

https://github.com/czproject/git-php