GIT und GITHUB Basisdoku

- https://rogerdudler.github.io/git-guide/index.de.html
- https://git-scm.com/docs/everyday
- https://github.com/yui/yui3/wiki/Set-Up-Your-Git-Environment
- https://www.thomas-krenn.com/de/wiki/Git_Grundbefehle

Basisfunktionen & lokale Instanz

- Arbeitskopie = die echten Dateien, dynamisch
- `git add` addiert den aktuellen Datenstand in den Index ("Stage")
- `git coomit` trägt den Datenstand in den HEAD ein (standardmässig in den "main" Branch)
- HEAD zeigt immer auf letzten Commit. Ein `git commit -m "Commit-Nachricht"` übernimmt den Status nach HEAD.

Initialisierung: leeres Repository erzeugen und die darin vorhandenen Dateien in den lokalen git Index aufnehmen

```
git init
git add .
```

Danach initialer Commit, um den aktuellen Stand im HEAD aufzunehmen

```
git commit -m "Commit-Nachricht"
```

Nachträgliches Ändern eines commit Textes

```
git commit –amend -m "xxxxx"
```

Remote Repos

"origin" ist das **eigene** Standard-remote-Repo (z.B. github, gitea). "upstream" ist das Team-Repo. Änderungen aus "origin" werden mit einem Pull Request für das "upstream" Repo angeboten. Von dort wird die Änderungen per "pull" geholt, falls sie akzeptiert wird.

Kontrolle der Remote Ziele "origin" und "upstream"

```
git remote -v

origin  git@github.com:thommierother/docs.git (fetch)
origin  git@github.com:thommierother/docs.git (push)
upstream        git@github.com:owncloud/docs.git (fetch)
upstream        git@github.com:owncloud/docs.git (push)
```

Nach dem Anlegen eines eigenen Repos auf github kann man das lokale Repo mit einem Repo auf github verknüpfen.

- origin = persönliches git Repo auf Github. Die kann ein privater Fork eines öffentlichen Repos auf github sein.
- upstream = Master repo, eigene commits landen dort per Pull Request

Aus https://www.theserverside.com/video/How-to-use-Gits-set-upstream-push-command :

Fix upstream branch errors with autoSetupRemote

If you constantly run into Git's fatal "Current branch has no upstream branch" error, and you find issuing the git push –set-upstream command constantly to be annoying, you can tell Git to use the set-upstream option automatically whenever you push a new branch to the server.

This is done by setting the push.autoSetupRemote option in any of Git's configuration files.

How to set push.autoSetupRemote

To tell Git to automatically create new branches in remote repositories upon a push, simply issue the following command: git config –global –add –bool **push.autoSetupRemote** true

With that configuration setting in place, you'll never have to push with the git push –set-upstream command ever again.

öffentliche Repos: Kollaboration

Öffentliche Repos werden zunächst geforkt, also eine Kopie im eigenen Repo-Speicher als "origin" angelegt. Lokale Änderungen werden zuerst nach origin übertragen. Danach werden sie per pull request (PR) zur Übernahme nach upstream "angemeldet".

- upstream = öffentliches Github Repo (z.B. https://github.com/owncloud/core).

Lokale Instanz: hinzufügen des entfernten "origin"

```
git remote add origin git@github.com:thommierother/oc-theme-nw2.git
git push -u origin master
```

Analog geht das Verbinden des öffentlichen Repos als "upstream":

```
git remote add upstream
''git@github.com:https://github.com/owncloud/docs.git''
```

Authentifizierung über access token

```
git remote set-url origin
```

```
https://username:token@github.com/username/repository.git
```

Synchronisation lokales Repo mit entfernten

pushen

`git push origin` Änderungen auf das eigene Repo oder auf den Fork **senden**

`git push upstream` Änderungen nach upstream **senden**

abholen

`git fetch origin` Änderungen vom eigenen Repo oder vom Fork eines öffentlichen Repos abholen

`git fetch upstream` Änderungen von upstream abholen

Github: entferntes Repo klonen

Lokale Kopie des aktuellen Arbeitsstandes auf github anlegen

`git clone /pfad/zum/repository`

`git commit -m "Commit-Nachricht` Änderungen hochladen

Update des Fork auf Github

Das geht über das **lokale** Repo:

Updates von Upstream holen

```
''$ git fetch upstream''
```

Zum master bzw. main wechseln

```
''git checkout master''
```

Master mit upstream mergen

```
''git merge upstream/master''
```

Danach push auf den fork setzen.

Branches

Branch erzeugen

```
git checkout -b iss53
```

Wo bin ich (aktueller Branch)

```
git branch
```

Alle Branches im lokalen und im remote Repo zeigen (remotes)

```
git branch -a

  main
* master
  remotes/origin/HEAD -> origin/main
  remotes/origin/main
  remotes/origin/master
```

Nur die remotes zeigen

```
git branch -r
```

Zum anderen Branch wechseln

```
git checkout master
```

Freier Wechsel zu Branch

```
git checkout [branchname]
```

Lokal erzeugten Branch im remote Repo verfügbar machen

```
git push origin master
```

synchronisiert in den Master Branch vom Remote Repo (origin)."origin" weist auf den (privaten) Fork eines Github-zentralen Repos.

Neuen Branch erstellen und zu diesem wechseln

```
git checkout -b feature_x
```

Branch löschen

```
git branch -d feature_x
```

Download eines bestimmten Branch

```
git checkout -b workstation-16.2.1 origin/workstation-16.2.1
```

danach

```
git pull
```

Oder:

```
git clone -b workstation-17.0.2
https://github.com/mkubecek/vmware-host-modules.git
```

Merge & update

```
git pull
```

Holt Änderungen von Remote (fetch) **und führt sie mit dem lokalen Stand zusammen** (merge). Bei Konflikten: Dateien manuell korrigieren und danach die geänderte Datei mit `git add [dateiname]` einbauen. Die Unterschiede sieht man mit

```
git diff <quell*branch*> <*ziel*branch>
```

Upstream nach fork mergen

Lokales Ziel

```
git checkout //master//
```

Welche branch von Upstream soll wohin geholt werden?

```
git pull git@github.com:owncloud/docs.git BRANCH_NAME
```

Danach commit, review und push auf den fork bei Github

Wenn etwas ganz schiefgeht

```
git checkout —
```

setzt die lokalen Änderungen auf den letzten HEAD Stand. Änderungen, die du bereits zum Index hinzugefügt hast, bleiben bestehen.

Hard Reset = Zurück auf den letzten Stand vom entfernten Repository:

`git fetch origin git reset –hard origin/master`

Pull requests

Pull requests let you tell others about changes you've pushed to a branch in a repository on GitHub. Once a pull request is opened, you can discuss and review the potential changes with collaborators and add follow-up commits before your changes are merged into the base branch.

https://www.digitalocean.com/community/tutorials/how-to-create-a-pull-request-on-github

Basis Konfiguration eines Repos

`git config -global user.name [name]`
 	
`git config -global user.email [email]`
 	
`git config -global core.editor [editor]`
	GIT User Konfiguration setzen

`git config -l`	**Konfiguration zeigen**
`git status`	**zeigt,ob eine Datei editiert wurde**
`git-diff and git-status`	**Show the working tree status**
`git commit`	**Änderungen einspielen**
`git checkout -b branch2`	**Wechsel zwischen Branches: Neuen Branch erstellen und dort hin wechseln**
`git-reset[1]`	**Reset current HEAD to the specified state**
`git-merge`	**Join two or more development histories together**
`git-rebase`	**Reapply commits on top of another base tip**
`git tag`	**Tags zeigen (z.B. einzelne Releases)**
`git log`	**Letzte Commits zeigen**
`git fetch <remote>`	**Objektstruktur runterladen**

Repo duplizieren

Create a bare clone of the repository.

$ git clone –bare https://github.com/exampleuser/old-repository.git

Mirror-push to the new repository.

$ cd old-repository.git $ git push –mirror https://github.com/exampleuser/new-repository.git

Remove the temporary local repository you created earlier.

$ cd .. $ rm -rf old-repository.git

Alte commits entfernen

Commits sichten

```
git log --oneline
```

Neuen Branch erzeugen, aber ohne history (–orphan)

```
git checkout --orphan tem_branch
```

Alle Files hinzufügen

```
git add -A
```

Inital commit im neuen Branch

```
git commit -am "Initial commit message"
```

Alten Branch löschen

```
git branch -D main
```

Den aktuellen Brach umbenennen

```
git branch -m main
<code>

Den "zurückgesetzten" main Branch aufs remote laden

<code>
git push -f origin main
```

Git mit SSH

Mehrere ssh Key benutzen:

Keys zum agent hinzufügen:

```
 <font inherit/monospace;;inherit;;#000000background-color:#ffffff;>ssh-add
~/.ssh/id_rsa3</font>
```

.ssh/config

```
 <font inherit/monospace;;inherit;;#000000background-color:#ffffff;>Host
gitea gitea.netzwissen.de</font>       ForwardAgent yes
```

```
#HostName gitea.netzwissen.de
# IdentityFile ~/.ssh/id_rsa_gitea
# User git

Host 192.168.72.12
IdentityFile ~/.ssh/id_rsa
User thommie

Host kakariki.netzwissen.de
HostName kakariki.netzwissen.de
IdentityFile ~/.ssh/id_rsa3
```

Im Repo selbst

.git/config

```
 <font inherit/monospace;;inherit;;#000000background-
color:#ffffff;>[core]</font>            repositoryformatversion = 0
       filemode = true
       bare = false
       logallrefupdates = true
**       sshCommand = ssh -i ~/.ssh/id_rsa**
[remote "origin"]
       url = ssh://git@gitea.netzwissen.de:2022/thommie/thommievault.git
       fetch = +refs/heads/*:refs/remotes/origin/*
[branch "main"]
       remote = origin
       merge = refs/heads/main
```

Gitea

Git with a cup of tea - A painless self-hosted Git service

https://gitea.io/en-us/

https://dl.gitea.io/gitea/xxx/gitea-xxx-linux-arm64

https://docs.gitea.io/en-us/upgrade-from-gitea/

Upgrade script: /etc/scripts/gitea_upgrade.sh