Git

#First Phase:

#Git global setup User name and email

git config --global user.name “Zigong Xu”

git config --global user.email [xu@physik.uni-kiel.de](mailto:xu@physik.uni-kiel.de)

git init

git remote add origin url.git

git add .

git commit –m ‘Initial commit’

git push –u origin master

# git generate local branch and push it to remote locally

Git branch New\_branch

Git checkout New\_branch

“Edit”

Git add New\_file or Git add --update : add the file already in tree

Git commit -m ‘New\_branch’

Git push -u origin New\_branch

:A new branch will be automatically generated

The

Git pull or git push to update and download

# git generate a local branch and link to the remote existed one

Git branch –av : to list all the local and remote branch

# assuming in remote already exist a branch named NN\_branch and I would like to have a one with same name.

Git branch NN\_branch

Git checkout NN\_branch

git branch --set-upstream-to origin/NN\_branch ; link the local with the remote one

git pull ; to update the local branch

git push –u origin NN\_branch

# also you can change the link to different branch.

#\_\_

git checkout -b dev origin/dev

# this command could releaze the function above and don’ t need to track stream or branch.; From

#---

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#git branch --set-upstream-to origin/NN\_branch

#: set the current branch to the remote origin/NN\_branch.

# You can change it in diferent branch

------------------------------------------------

How to delete local branch and remote branch:

Git branch –av ; list all the information about the branch and remote branch.

Git branch –D name\_of\_branch ; Note: captail letter, or other way see the link

#

Git push origin –delete name\_of\_remote\_branch ; Delete remote branch

Cf: <https://www.educative.io/edpresso/how-to-delete-remote-branches-in-git>

Git remote remove origin

# break the link between local and remote

git remote set-url origin git://new.url.here

# change the url of remote git repository

git remote add origin yourRemoteUrl-or-SSHUrl

#git add the remote

git push -u origin master

# push the local branch master to the remote.

#git clone

git clone git://\*\*\*\*.git

#or

git clone <http://www.kernel.org/pub/scm/git/git.git>

git clone robin.hu @ http://www.kernel.org/pub/scm/git/git.git

# if you need the name and password;

git clone -b dev\_jk http://10.1.1.11/service/tmall-service.git

# specific the certain branch

Reference:

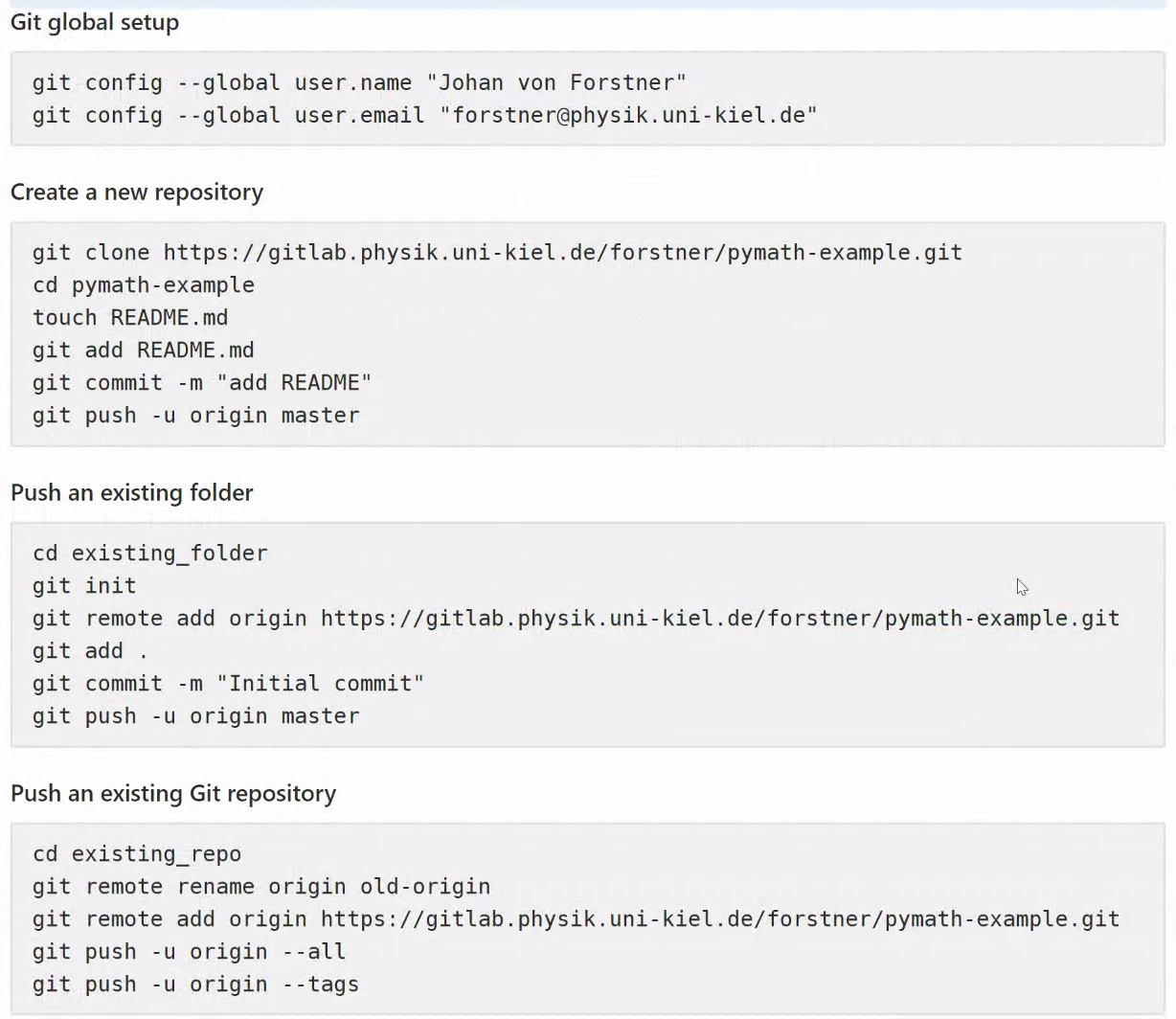
<https://www.freecodecamp.org/forum/t/push-a-new-local-branch-to-a-remote-git-repository-and-track-it-too/13222>

<https://github.com/Kunena/Kunena-Forum/wiki/Create-a-new-branch-with-git-and-manage-branches>

<https://www.liaoxuefeng.com/wiki/896043488029600> A quite good websit

<https://blog.csdn.net/wangjia55/article/details/8818845>

https://blog.csdn.net/weixin\_39800144/article/details/78205617



Advanced

.gitignore

# could ignore the files you don’t want to commit

.gitlab-ci.yml

# run some default check or run in background

# even could compile and generate some pdf file

# In github, similar things finished in ACTION. Just try it and you will know

.git hooks file

-could do some function automatcially when they find you do certain movement.

hook，直译过来是“钩子”，通常是用于在某事件发生或者完成后添加自定义的动态事件/任务。在使用 git 时，我们也可以通过 git hook 来完成一些自动测试、集成、构建等流程工作。如果 git hook 执行失败将终止后续操作。

**有哪些可用的 git hook**

在项目的 .git/hooks 目录中会有很多 hook-name.sample 的文件，提供了常见 hook 的 shell 脚本模板，当然我们也可以使用 Python、Ruby 等其它脚本语言编写 hook，不过不要忘记在脚本首行加上 shebang 声明，并赋予可执行权限。

根据 hook 执行的位置，可以分为“客户端 hook”和“服务器端 hook”，客户端 hook 在开发者本地环境运行，服务器端 hook 则运行在项目的托管服务器上，通常是在接收提交时触发。

**客户端 hook**

**git commit 相关 hook**

* pre-commit：提交前执行，可以使用 git commit --no-verify 命令跳过该 hook。
* prepare-commit-msg：在启动提交信息编辑器之前执行，用于定制提交时出现信息。可以结合提交模板使用，动态地插入信息。
* commit-msg：
* post-commit：提交过程完成后运行的 hook，不接收任何参数。

**git am 相关 hook**

git am 用于应用（apply）来自邮箱（mailbox）的 patch，am 正是 “apply mailbox” 的缩写。am 流程会检测 3 个 hook：

* pre-applypatch：运行于应用补丁**后**、产生 commit 前。因此更合理的命名可能应该是 post-applypatch-pre-commit。
* applypatch-msg：它接收单个参数：包含请求合并信息的临时文件的名字。 如果脚本返回非零值，Git 将放弃该补丁。 你可以用该脚本来确保提交信息符合格式，或直接用脚本修正格式错误。
* post-applypatch：应用补丁并产生提交之后。

**其它客户端 hook**

* pre-rebase：运行于 git rebase 之前，官方的例子中是用于禁止对已推送的 commit 进行 rebase 操作，这样可以防止于线上的代码库发生冲突。
* post-checkout：checkout 命令执行完成之后会调用的 hook，通常用于自动调整运行环境。
* post-merge：运行于合并操作完成之后。
* pre-auto-gc：在运行 git gc --auto 命令前执行，可以用于提醒即将进行垃圾回收，或者现在是否应当进行垃圾回收。
* post-rewrite：运行于重写提交信息的命令之前，比如 git commit --amend 之前。
* pre-push：在 push 操作前执行。

**服务器端 hook**

* pre-receive：接收到来自客户端的推送时最先调用的 hook，可以用户阻止“non-fast-forward” 的更新，或者进行文件访问控制。
* update：类似 pre-receive，不过在同时推送多个分支时 pre-receive 仅仅运行一次，update 会每个分支执行一次。
* post-update：运行于整个接收服务完成之后。

-Format code

vscodium/ pycharm/ pytest