git common commands

document author: Yongjie Lyu Yongjie.L@outlook.com

1.add/commit

```
$ git add readme.txt
$ git commit -m " append GPL "        //本次修改的注释:append GPL
```

（1）git add 实际上就是把文件修改添加到暂存区；
（2）git commit 实际上就是把暂存区的所有内容提交到当前分支。
（3）可以进行多次的 add，之后进行一次 commit

2.config

```
$ git config --list
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
user.name=LYJ
user.email=lvyj75@chinaunicom
...
```

3.status

```
$ git status      //掌握仓库当前的状态
On branch master
nothing to commit, working tree clean
```

4.diff

```
$ git diff learngit.txt      //查看具体修改内容
diff --git a/learngit.txt b/learngit.txt
index 63e1316..410438a 100644
--- a/learngit.txt
+++ b/learngit.txt
@@ -1,3 +1,3 @@
 Git is a distributed version control system.
 Git is free software.
-nice!!!(2.0)
\ No newline at end of file
+nice!!!(3.0)
\ No newline at end of file
5.log
```

5. $ log

```
$git log                              //命令显示从最近到最远的提交日志
$ git log --pretty=oneline            //简略版本
$ git log --graph                     //分支合并图
$ git log --graph --pretty=oneline --abbrev-commit     //看分支的合并情况（图示）
```

6.reset

$ git reset --hard HEAD^ //回退到上一个版本
$ git reset --hard HEAD^^ //回退到上上一个版本
$ git reset --hard 1094a //回退到一个特定版本（版本号 1094a…）
$ git reset HEAD LICENSE.txt //将暂存区的修改回退到工作区
git reset 命令既可以回退版本，也可以把暂存区的修改回退到工作区。当我们用 HEAD 时，表示最新的版本。

7.reflog

$ git reflog //记录命令（可以重新返回修改已经 reset 了的版本）

8.工作区与暂存区

（1）工作区（Working Directory）：就是你在电脑里能看到的目录，比如我的 learngit 文件夹就是一个工作区；
（2）版本库（Repository）工作区有一个隐藏目录.git，这个不算工作区，而是 Git 的版本库。
Git 的版本库里存了很多东西，其中最重要的就是称为 stage（或者叫 index）的暂存区，还有 Git 为我们自动创建的第一个分支 master，以及指向 master 的一个指针叫 HEAD。
（3）git 专注于修改，而不是文件!
当对一个文件做出修改后，git 可以对他进行持续实时的追踪，但是如果不将该文件 add 进暂存区中的话，即便 commit 也不会将这次修改合并到分支上去

9.checkout

$ git checkout -- readme.txt //把 readme.txt 文件在工作区的修改全部撤销
（注意“--”命令很重要，去掉就是切换分支!!! 并且注意：“--”符号两端需要追加空格）
有不同的撤销情况：
(1)readme.txt 自修改后还没有被放到暂存区，现在，撤销修改就回到和版本库一模一样的状态；
(2)readme.txt 已经添加到暂存区后，又作了修改，现在，撤销修改就回到添加到暂存区后的状态。
(3)在工作区将文件删除后，也可以使用该命令撤销删除
总之，就是让这个文件回到最近一次 git commit 或 git add 时的状态。

10．push

$ git push -u origin master //第一次提交
使用-u 命令是第一次推送 master 分支时，Git 不但会把本地的 master 分支内容推送的远程新的 master 分支，还会把本地的 master 分支和远程的 master 分支关联起来，在以后的推送或者拉取时就可以简化命令。
$ git push origin master //后续正常提交
orgin 远程库名称；master 指定的本地分支名称

11.clone

$ git clone git@github.com:ddrangers/gitskills.git //将远端代码克隆到本地

12.分支

主分支,master 分支： 指向最新的提交

HEAD：指向当前分支

(1) checkout&branch

$ git switch –c dev //创建并切换到新的 dev 分支
$ git switch master //切换分支
$ git branch dev **//创建分支**
$ git checkout -b dev //创建分支 dev 并且切换到 dev 分支
$ git checkout dev //切换分支

(2) branch

$ git branch //查看当前分支

(3) merge

$ git merge dev //将 dev 分支合并到 master 分支上
$ git merge --no-ff -m " noff " dev //使用 noff 方式将将 dev 分支合并到 master 分支上
加上--no-ff 参数使合并后的历史有分支，能看出来曾经做过合并

(4) branch –d

$ git branch -d dev //删除 dev 分支

(5) cherry-pick

之前的 master 分支有 bug，现在已经通过"[issue-101 4c805e2] fix bug 101 分支"修复，但是在修复之前就出现了分支 dev，所以现在需要把该修复分支应用在目前的 dev 分支上

修复分支代码：

```
$ git add readme.txt
$ git commit -m "fix bug 101"
[issue-101 4c805e2] fix bug 101
 1 file changed, 1 insertion(+), 1 deletion(-)
```

$ git cherry-pick 4c805e2 //git 自动对当前所在分支进行一次提交，使我们能够复制一个特定的提交到当前分支（避免重复工作）

13.分支策略

master 分支：主分支时刻与远程保持同步，保持稳定仅用来发布新版本；
dev 分支：开发分支，团队成员都在 dev 分支上开发，需要和远程同步；
bug 分支：只用于在本地修复 bug，不需要推送远端
feature 分支：取决于是否是小组联合开发

14.stash

$ git stash //把当前工作现场"储藏"起来，等以后恢复现场后继续工作
$ git stash list //显示存储的工作线现场
$ git stash apply //恢复现场，但不删除 stash
$ git stash pop //恢复现场，并且删除 stash
$ git stash apply stash@{0} //恢复指定的 stash

有时候正在工作区代码写了一半的时候，需要修复 bug，就把这个工作区先储存起来，去创建分支修复 bug，完成之后回到原分支恢复原工作现场

15.多人协作

(1) $ git remote //查看远程库信息

(2) $ git remote –v //查看远程库详细信息

(3) $ git branch --set-upstream-to=origin/dev dev

//指定本地 dev 分支与远程 origin/dev 分支的链接

(4) 多人协作工作模式:

a) 首先，可以试图用 `git push origin <branch-name>` 推送自己的修改;
b) 如果推送失败，则因为远程分支比你的本地更新，需要先用 `git pull` 然后尝试合并分支;
c) 如果合并分支有冲突，则解决冲突，并在本地提交;
d) 没有冲突或者解决掉冲突后，再用 `git push origin <branch-name>` 推送就能成功!

16.标签

$ git tag <name> //打一个新标签（需提前切换到打标签的分支上，默认打在最新提交的 commit 上）

$ git tag //查看所有标签

$ git tag <name> f52c633 //为之前的 commit 打标签，“f52c633”是之前的一次 commit id

$ git show <tagname> //查看标签信息

$ git tag -d v0.1 //删除标签

$ git push origin <tagname> //推送某个标签到远程库(标签是默认在本地存储的，不会自动推送到远程)

删除远程库标签