目录

[git基本用法 1](#_Toc477212220)

[基础入门 1](#_Toc477212221)

[必会命令： 1](#_Toc477212222)

[工作区与暂存区 2](#_Toc477212223)

[工作区 2](#_Toc477212224)

[版本库 2](#_Toc477212225)

**参考：**[**http://www.cnblogs.com/alex3714/articles/5930846.html**](http://www.cnblogs.com/alex3714/articles/5930846.html)

# git基本用法

# 基础入门

## 必会命令：

1. 初始化

git init

1. 查看状态

git status

git status -s

1. 添加修改或添加新文件

git add file\_name

git add . #添加所有更新或新增的文件

1. commit

git commit –m “say something”

1. 提交到仓库

git push origin master

1. 对比

git diff

1. 查看日志

git log

git log –pretty=oneline #每行一条日志

1. 回退到上一版本

git reset --hard HEAD^

1. 回退到指定版本

git reset --hard commit\_ID #只需要记得前四五位即可

1. 查看git 的每次命令

有助于找回commitID

git reflog

1. 放弃工作区的修改

git checkout -- file\_name

1. 将add 到stage的内容回滚

git reset HEAD filename

git checkout -- filename #丢弃工作区的修改

1. 删除文件

（文件已经推送到版本库，即已经commit）

rm filename

git rm filename

git commit #这时版本库已经没有了，无法回滚

1. 删除错误后回滚

rm filename

git checkout -- filename

## 工作区与暂存区

### 工作区

就是电脑里能看到的目录，比如study\_git就是一个工作区

[root@localhost devops]# tree study\_git/

study\_git/

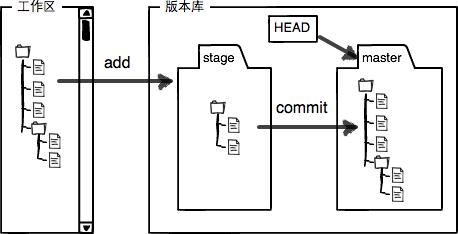
└── first\_git\_file.txt

0 directories, 1 file

### 版本库

在工作区里有一个隐藏目录.git,这个就是版本库

版本库里最重要的是stage（or index）,可以称为暂存区，还有git为我们创建的第一个分支master,以及指向master的指针HEAD



1. git add 将文件添加进stage
2. git commit 将暂存区的所有内容添加进当前分支

添加一个readme.txt文件并写入内容

git add .

[root@localhost study\_git]# git status

# On branch master

# Untracked files:

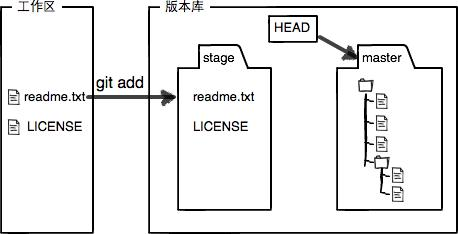
# (use "git add <file>..." to include in what will be committed)

#

# readme.txt

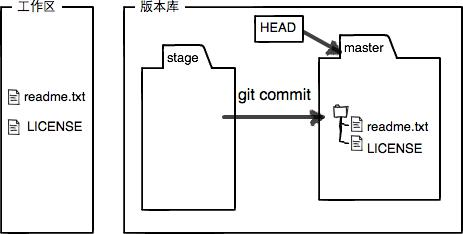
nothing added to commit but untracked files present (use "git add" to track)

stage的状态如下图



git commit 后

暂存区的图示如下



## 远程管理

1. 将本地代码推到远程仓库

git remote add origin [git@git.oschina.net:yiday/crazyEye.git](mailto:git@git.oschina.net:yiday/crazyEye.git)

git push –u origin master # git push –f 强推

origin:远程分支

master:本地分支

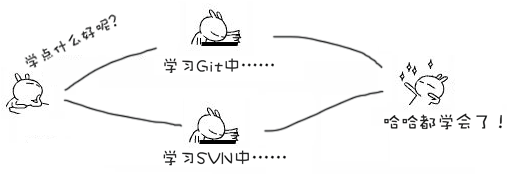
1. 将远程仓库克隆到本地

git clone [git@git.oschina.net:yiday/crazyEye.git](mailto:git@git.oschina.net:yiday/crazyEye.git)

git clone --depth=1 指定获取最后1个版本

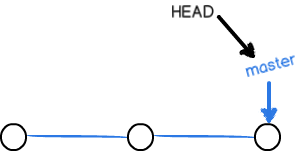
如果不加--depth 则获取所有的历史版本，有些开源项目数据仓库很大，造成不必须的内容下载

## 分支管理



### 创建与合并分支

1. 默认只用master分支，用master指向最新的提交，再用HEAD指向master，就能确定当前分支，以及当前分支的提交点，如图



每次提交，master就会向前移动一步

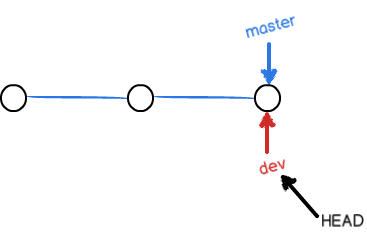
1. 创建一个叫dev的分支，并将HEAD移动到dev

git checkout –b dev

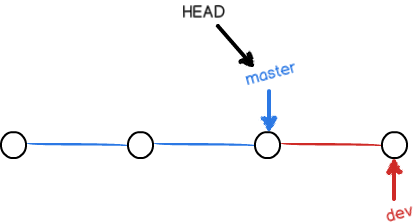
相当于：

git branch dev

git checkout dev



在dev上进行开发



1. 合并dev与master 分支的内容

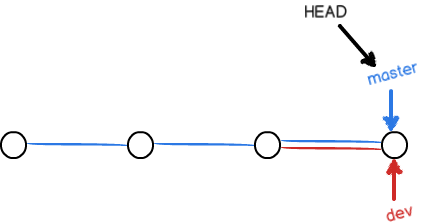
为了合作开发，必须合并分支到master

1. 先将HEAD切回master

git checkout master

1. 合并分支

git merge dev



1. 删除dev分支

git branch –d dev

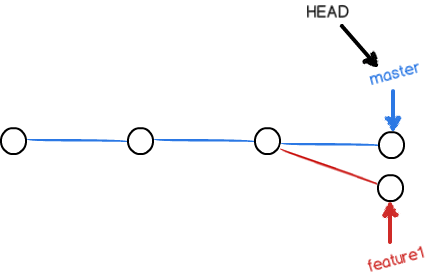
1. 列出所有分支

git branch

### 解决冲突

1. 其他分支与master同时修改了同一个文件，在merge时就有可能产生冲突

如行数同的情况下，那么就要手动合并



1. 无法’快速合并’,只能通过手工修改合并

git merge feature1

显示冲突的文件

[root@localhost crazyEye]# cat readme.md

Python 开发的监控系统

creating a new branch is quick.

<<<<<<< HEAD

add this line from master

=======

added this line from branch feature 1

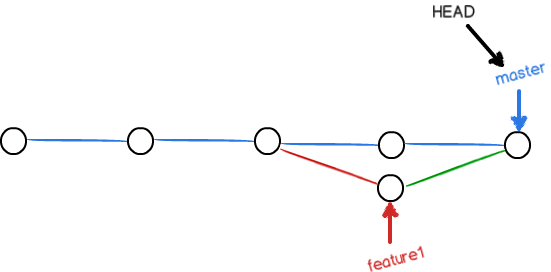
>>>>>>> feature1

NOTE: Git用<<<<<<<，=======，>>>>>>>标记出不同分支的内容

1. 修改冲突并提交

git add readme.md

git commit –m “conflict fixed”



1. 用图未来查看合并情况

[root@localhost crazyEye]# git log --graph --pretty=oneline

\* 2e92bf6c97ad822d6bdcfa22ff2aca5ac2aff973 conflict fixed

|\

| \* 6c98409a104dba52189490e6e709c2732a84285a add feature

\* | 7306f66286dddb83976702b7afc015cf066343e7 master update

|/

\* 988f7b03a9c2c1b7117e081b2cebbd90acf7ddca branch test

\* 0a3048bb1a1c6000ba8e51f013d63dad01451b17 add main.py

\* 07597500c3ce3bf9014a692e4f0c9b7ec81ccfd8 add readme.md

### 分支策略

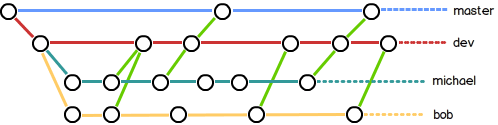
在实际开发中，我们应该按照几个基本原则进行分支管理：

首先，master分支应该是非常稳定的，也就是仅用来发布新版本，平时不能在上面干活；

那在哪干活呢？干活都在dev分支上，也就是说，dev分支是不稳定的，到某个时候，比如1.0版本发布时，再把dev分支合并到master上，在master分支发布1.0版本；

你和你的小伙伴们每个人都在dev分支上干活，每个人都有自己的分支，时不时地往dev分支上合并就可以了。

所以，团队合作的分支看起来就像这样：



### BUG分支

## 基本配置

### 忽略文件

添加.gitignore

\*.a       # 忽略所有 .a 结尾的文件

!lib.a    # 但 lib.a 除外

/TODO     # 仅仅忽略项目根目录下的 TODO 文件，不包括 subdir/TODO

build/    # 忽略 build/ 目录下的所有文件

doc/\*.txt # 会忽略 doc/notes.txt 但不包括 doc/server/arch.txt

# 进阶中级

## 必会命令

1. 获取最后一次提交的时间戳

git log --pretty=format:%ct --quiet -1 HEAD