

GIT

git的安装

windows

直接官网下载

linux

```
sudo apt-get/yum install git
```

从源码安装

```
./config
```

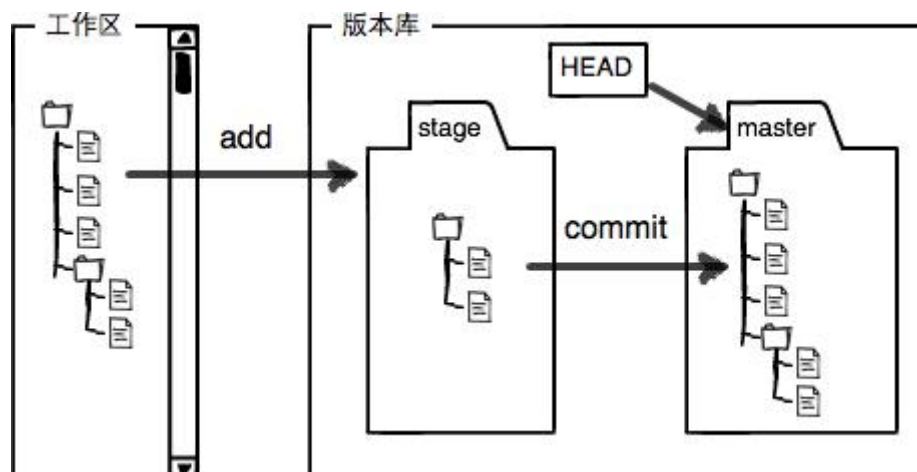
```
make
```

```
sudo make install
```

工作区和暂存区

git 和其他版本控制系统不同就在于就是有暂存区的概念

- **工作区**：工作区就是运行 `git init` 的文件夹
- **版本库**：工作区有个隐藏的 `.git` 文件夹，它就是版本库
- **暂存区**：git 的版本库里存了很多东西，最重要的是成为 **stage**（或 **index**）的暂存区，还有 git 自动创建的第一个分支 **master**，以及指向 **master** 的指针 **HEAD**



创建仓库

1. 新建一个文件夹
2. 进入文件夹
3. 执行 `git init`

提交新文件到库中

`git add 文件`: 将文件从工作区添加到暂存区

`git commit -m "说明"`: 将暂存区的所有内容提交到当前分支

即需要提交的文件修改通通放到暂存区, 然后, 一次性提交暂存区的所有修改。

可以使用 `git status` 查看当前仓库的文件状态

提交修改文件到库中

`git add 文件`

`git commit -m "说明"`

`git diff`: 查看暂存区和工作区文件的区别

`git diff HEAD -- test.txt`: 查看未提交的文件 (包括已经 `add` 的修改和未 `add` 的修改) 和版本库里最新版本的区别

版本回退

`git log` 查询提交的历史记录, 其结果为从按时间从近到远显示, 结果如下:

```
Davidswin@DESKTOP-UK7NDU9 MINGW64 /d/notes (master)
$ git log
commit a0b8554a21f94006d0487d4c38b1b57f258ccb50 (HEAD -> master)
Author: davidswin <1440290539@qq.com>
Date:   Fri Jan 24 14:08:22 2020 +0800

    新建了git.md用于保存git学习笔记

commit aa2dbd7d5df1240d148d560743d68884ebfff4ce
Author: davidswin <1440290539@qq.com>
Date:   Fri Jan 24 00:03:57 2020 +0800

    alter the mysql.md

commit 47dbbc479d006baa35978082ddf6ba88bf43069c
Author: davidswin <1440290539@qq.com>
Date:   Thu Jan 23 23:57:00 2020 +0800

    wrote a mysql.md file
```

可以加上 `--pretty=oneline` 参数, 使得结果在一行

```
Davidswin@DESKTOP-UK7NDU9 MINGW64 /d/notes (master)
$ git log --pretty=oneline
a0b8554a21f94006d0487d4c38b1b57f258ccb50 (HEAD -> master) 新建了git.md用于保存git学习笔记
aa2dbd7d5df1240d148d560743d68884ebfff4ce alter the mysql.md
47dbbc479d006baa35978082ddf6ba88bf43069c wrote a mysql.md file
```

图中一串十六进制的数字是 commit id, 标志每一次提交

版本回退操作方法

- `git reset --hard HEAD^` 该命令返回上一个版本如果是两个 `^` 则返回上上一个版本, 如果是 `~100` 则返回100版本前
- `git reset --hard commit_id` 根据提交id返回到指定的版本, 版本号不用写全, 写前五位就可以
- `git reflog` 用于查看命令历史, 使得无论什么时候都可以查询到库的版本信息

```
Davidswin@DESKTOP-UK7NDU9 MINGW64 /d/notes (master)
$ git reflog
f88e245 (HEAD -> master) HEAD@{0}: commit: 添加版本回退相关内容
f9e9f92 HEAD@{1}: reset: moving to f9e9f
1ae6b8d HEAD@{2}: reset: moving to HEAD^
f9e9f92 HEAD@{3}: commit: 添加test.txt文件用于测试
1ae6b8d HEAD@{4}: commit: 添加版本回退相关的笔记
a0b8554 HEAD@{5}: reset: moving to HEAD^
2d0f909 HEAD@{6}: commit: 新建测试文件test.txt
a0b8554 HEAD@{7}: commit: 新建了git.md用于保存git学习笔记
aa2dbd7 HEAD@{8}: commit: alter the mysql.md
47dbbc4 HEAD@{9}: commit (initial): wrote a mysql.md file
```

管理修改

git之所以更优秀是因为它跟踪的是**修改**而不是**文件**

什么是修改?

添加新文件, 删除文件, 新增若干行, 删除若干行, 修改若干个字符

添加修改

每次修改, 如果不用 `git add` 到暂存区, 那就不会加入到 `commit` 中。

撤销修改

- `git checkout -- test.txt` : 把 test.txt 文件在工作区的修改全部撤销:
 - 情况一: test.txt自修改后还未add到暂存区, 现在撤销修改回到和版本库一模一样的状态
 - 情况二: test.txt已经添加到暂存区, 又作了修改, 现在撤销修改回到添加到暂存区后的状态

总之就是让这个文件回到最后一次 `git commit` 或者 `git add` 时的状态

命令中 `--` 很重要, 如果没有, 该命令的意思为切换到另一个分支, 在下面的内容将会做出分支切换的说明

- `git reset HEAD test.txt`: 把test.txt文件已经添加到暂存区的修改撤销掉, 回到最近一次 add 前, 如果继续撤销修改, `git checkout -- test.txt` 命令
- 假设已经将修改提交到了本地版本库中, 可以用 `git reset --hard HEAD^` 回退到上一个版本, 前提是没有推送到远程库

删除文件

以Linux系统为例子

1. 执行文件删除命令 `rm test.txt`

- 确实要从版本库中删除该文件

`git rm test.txt` `git commit` 说明

- 误删, 且需要恢复

- 还未 add: `git checkout -- test.txt` 撤销工作区的修改

- 已经 add:

`git reset HEAD test.txt`

`git checkout -- test.txt`

远程仓库

Git是分布式版本控制系统，同一个Git仓库，可以分布到不同的机器上。怎么分布呢？最早，肯定只有一台机器有一个原始版本库，此后，别的机器可以“克隆”这个原始版本库，而且每台机器的版本库其实都是一样的，并没有主次之分。

1. 创建SSH Key

```
ssh-keygen -t rsa -C "1440290539@qq.com"
```

在用户目录下：

id_rsa：私钥

id_rsa.pub：公钥

2. 登录GitHub，打开“settings”，“SSH and GPG keys”页面

添加远程仓库

1. 将远程仓库和本地仓库关联

- `git remote add origin git@github.com:heresun/myNotes.git`

- 把本地内容推送到远程库

```
git push -u origin master
```

`git push` 实际上是把当前分支master推送到远程，由于远程仓库是空的，第一次推送master时加上了 `-u` 参数，这样git不但把本地的master分支内容推送到远程master分支，还会把二者关联起来，在以后的 `push` 和 `pull` 时就可以简化命令

自此以后就可以用如下命令master内容到远程仓库

```
git push origin master
```

此时我在github创建仓库时添加了README.md文件，但是这个文件在我的本地库中没有，在运行 `git push -u origin master` 时发生了如下错误

```
To github.com:heresun/myNotes.git
! [rejected]        master -> master (fetch first)
error: failed to push some refs to 'git@github.com:heresun/myNotes.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

错误要求将本地master分支与远程master分支合并,命令如下

```
git pull --rebase origin master
```

`git pull` = `git fetch` + `git merge`

2. 从远程仓库克隆

当我们从零开发，最好的方式是先创建远程库再从远程库克隆

```
git clone git@github.com:heresun/notes.git
```

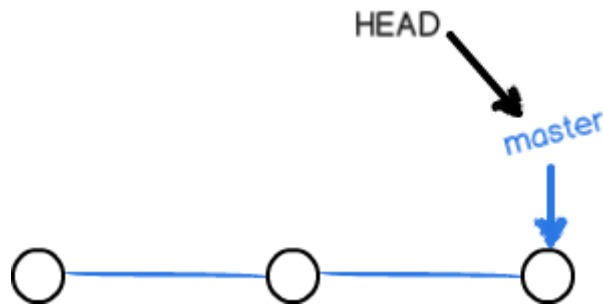
3. 查看远程仓库信息

```
git remote -v
```

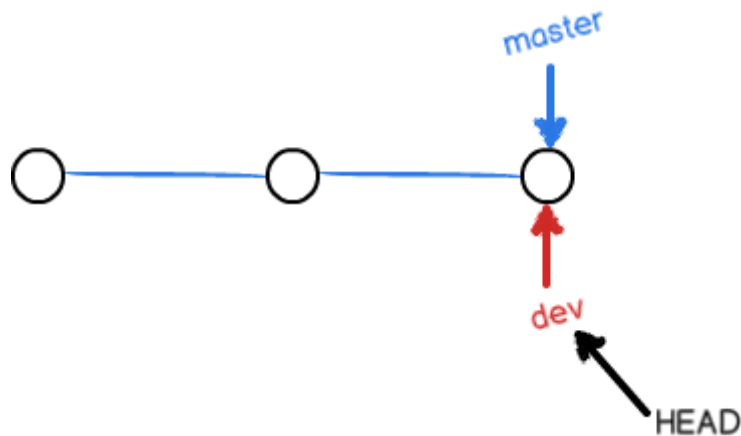
管理分支

项目组每个人都可以创建一个自己的分支，每个人在自己的分支上进行开发，可以在自己的分支上任意提交和回滚，等到自己的分支上开发完成后再将其合并到主分支上，自己的分支对别人来说是透明的，git的分支创建和切换是非常快的。

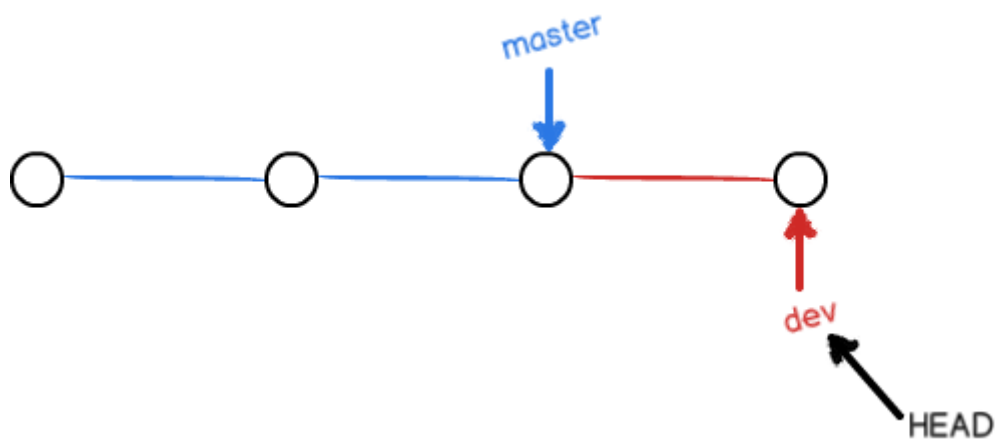
git把每次提交串成一个时间线，一个时间线就是一个分支，HEAD 严格来说不是指向提交，而是指向分支，如



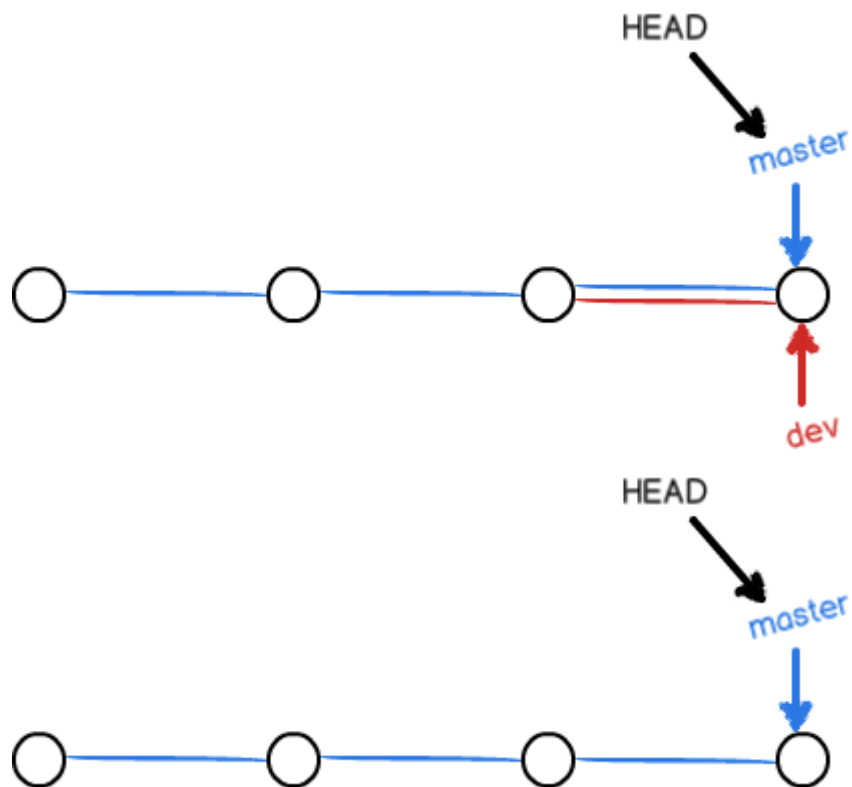
如上，master才是指向分支的实际指针，每次提交一次，master指针就向前一步，此时创建一个新的分支 `git branch dev` ,并且切换到该分支 `git checkout dev`



此时HEAD指向dev，以上两条命令可以合并如下：`git checkout -b dev`，意思为创建并切换到dev分支。此时对工作区的修改和提交就是在dev分支上的了，master分支不变



假如在dev分支上的工作完成了，可以将其合并到master上，即直接把master指针指向dev的当前提交，所以git的合并就是更改指针，很快



命令总结

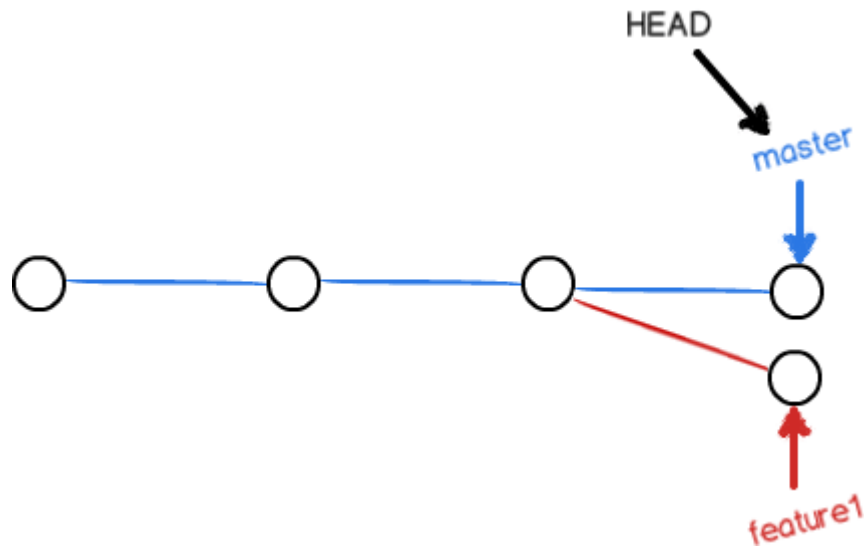
- `git branch dev`: 创建一个dev分支
- `git checkout dev`: 切换到dev分支
- `git switch dev`: 同上
- `git checkout -b dev`: 创建并切换到dev分支
- `git switch -c dev`: 同上

切换分支前必须把当前分支上进行的修改提交

- `git merge dev`: 合并dev分支
- `git branch -d dev`: 删除dev分支
- `git branch`: 查看当前分支

解决冲突

假设有master和feature1两个分支，此时两个分支各自有新的提交，如下图

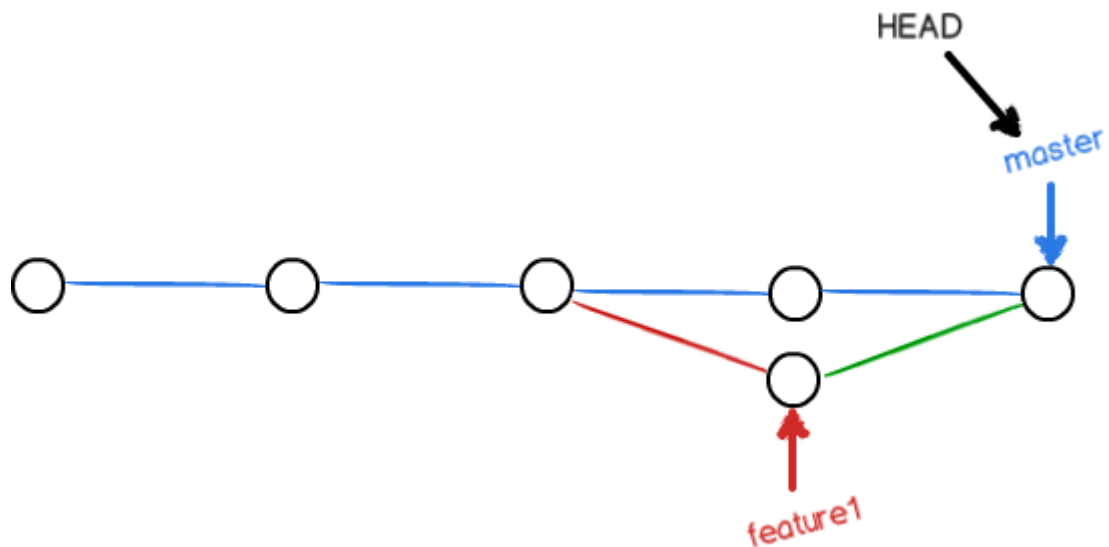


此时，git无法通过 `git merge feature1` 进行快速合并，强行用这种方式合并将会有冲突，如下

```
$ git merge dev
Auto-merging test.txt
CONFLICT (content): Merge conflict in test.txt
Automatic merge failed; fix conflicts and then commit the result.
```

提示test.txt这个文件存在冲突，必须手动解决后在提交

解决冲突后，重新在master分支提交，则变为如下图



feature1分支开发完成后，即可删除

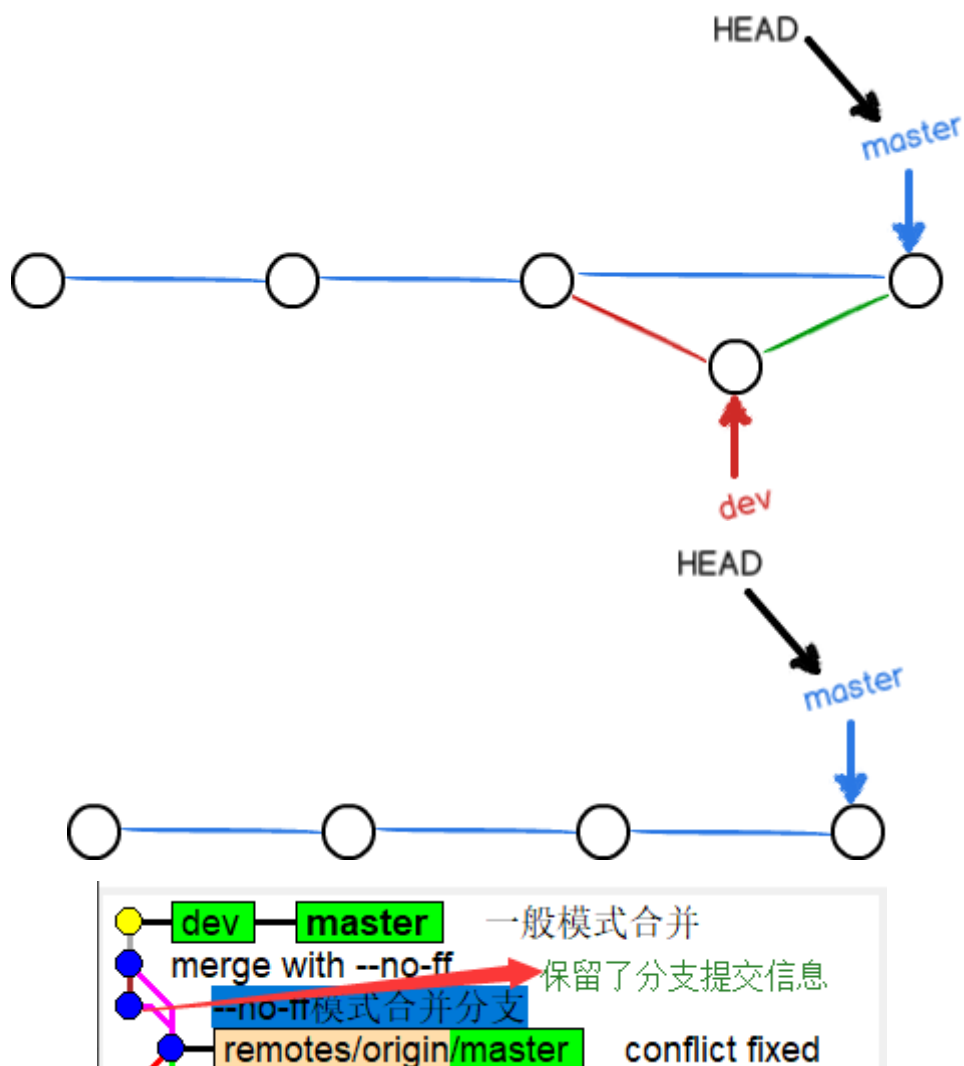
分支管理策略

通常情况下在合并分支时git会尽可能使用 `Fast Forward` 模式，在这种模式下，删除分支后，会丢掉分支信息

如果强制禁用该模式，git就会在merge时生成一个新的commit，这样从分支历史上可以看出合并信息

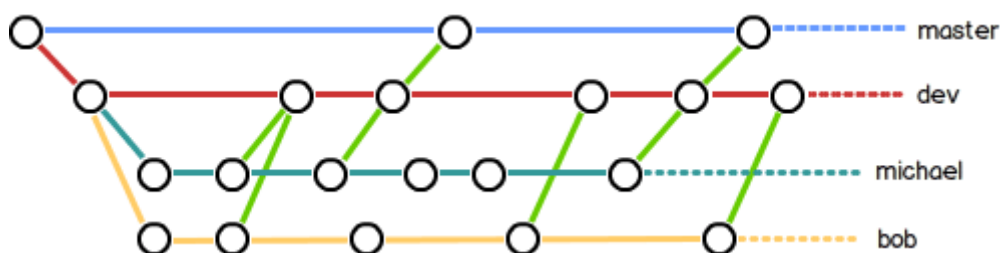
```
git merge --no-ff -m "合并提交说明" dev
```

上图时使用了 `--no-ff` 方式合并的，下图是fast forward模式



在实际开发中按照几个基本原则进行分支管理

1. master分支应该是非常稳定，仅仅用于发布新版本
2. 在dev分支上干活，如下图



bug分支与现场保存

假如现在master分支出现bug，需要我修复该bug，但是我当前的工作还未做完，预计还要很久，我的工作分支还有很多没有add和commit的修改，我该怎么办？

这是就可以将我的工作现场保存起来，去master分支修复bug，假如这个bug的代号为issue-101，那我就可以从master分支创建一个issue-101的bug分支，修复完成后再将其合并到master分支，再将issue-101分支删除，最后回到我的工作分支，恢复我之前的工作现场

具体命令如下：

- `git stash`: 在我的工作分支运行该命令，保存工作现场,如果此时运行 `git status` 会发现现在的工作区是干净的
- `git switch master`: 切换到master分支
- `git switch -c issue-101`: 创建bug分支

- 修复bug, 收尾工作, 切换到我的工作分支
- `git stash pop`: 恢复工作现场并将保存的备份删除

其他命令:

- `git stashlist`: 查看保存的工作现场
- `git stash apply` `git stash drop`: 这两条命令的最终效果等同于 `git stash pop`

思考

如果master存在bug那么dev分支是否也存在相同的bug, 答案是肯定的, 难道还需要在dev分支上重修手动修复相同的bug么? 修复当然是必须的, 但不需要手动

git提供了 `cherry-pick` 这个命令, 可以通过它复制一个特定的提交到当前分支, 假设在issue-101分支的提交id为4c805e2,那么在dev分支上就可以进行如下操作:

```
$ git cherry-pick 4c805e2
```

运行完这个命令后, git会自动给dev分支做一次提交, 虽然这两个提交改动相同, 但是是两个不同的提交

当然, 也可以在dev分支上修复bug, 然后用上面的命令将修复bug的提交复制到master分支上

Feature分支

开发一个新功能最好新建一个分支, 一般称其为feature分支, 这种分支的变动性是几极大的

比如: 为当前的项目添加代号为shiro的功能

1. `git switch -c shiro`

2. 开发

此时接到通知, 新功能取消, 目前该分支还未合并, 但是该功能包含公司机密, 需要就地销毁

- `git branch -d shiro` 无法删除该分支, 因为还未合并, 此时用 `git branch -D shiro` 强制删除

多人协作

推送分支

```
git push origin dev
```

本地分支如果不推送到远程, 对其他人是不可见的

但是, 并不是一定要把本地分支往远程推送, 那么, 哪些分支需要推送, 哪些不需要呢?

- `master` 分支是主分支, 因此要时刻与远程同步;
- `dev` 分支是开发分支, 团队所有成员都需要在上面工作, 所以也需要与远程同步;
- bug分支只用于在本地修复bug, 就没必要推到远程了, 除非老板要看看你每周到底修复了几个bug;
- feature分支是否推到远程, 取决于你是否和你的小伙伴合作在上面开发。

抓取分支

将本地分支与远程分支建立联系

- `git switch -c dev origin/dev`: 创建分支的同时与远程分支建立联系

- `git branch --set-upstream-to=origin/dev dev`: 将本地已存在的分支显式与远程分支建立联系

多人协作的模式

1. 首先, 可以试图用 `git push origin dev` 推送自己的修改;
2. 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` 试图合并;
3. 如果合并有冲突, 则解决冲突, 并在本地提交;
4. 没有冲突或者解决掉冲突后, 再用 `git push origin dev` 推送就能成功!

如果 `git pull` 提示 `no tracking information`, 则说明本地分支和远程分支的链接关系没有创建, 用命令 `git branch --set-upstream-to=origin/dev dev`。

变基 rebase

把分叉的提交历史整理成一条直线, 使其看上去更加直观, 缺点是本地的分叉提交已经被修改了

rebase可以把本地未push的分叉提交整理成一条直线

用法:

- `git pull --rebase`
- `git rebase`

标签管理

发布一个版本时, 通常下载版本库中打一个标签, 这样就唯一确定了打标签时刻的版本, 将来去某个标签的版本就是把那个打了标签的历史版本取出, 因此, 标签可以看作时版本库的一个快照。

但是, git的标签是指向某个commit的指针, 这个类似于分支, 但不同的是分支可以移动, 标签不可以, 但是已经有commit了, 既可以通过commit id找到, 为什么还需要标签? 这是因为标签是指向某个commit的有意义的名字, 容易记住。

创建标签

- `git tag version1.0`
默认标签是打在最新提交的commit上, 即HEAD上
- `git tag version1.0 f53c633`
在f53c6333这个提交上打一个名为 version1.0的标签
- `git tag -a version1.0 -m "标签说明" commit_id`: 创建带有说明的标签
- `git tag` 查看标签
标签不是按时间顺序排列的, 是按字母顺序的
- `git show tag_name`
查看标签名为tag_name的标签的详细信息

操作标签

- `git tag -d tag_name`: 删除本地标签
- `git push origin tag_name`: 推送标签到远程
`git push origin --tags`: 将本地标签全部推送到远程

- 删除远程标签：

1. 先从本地删除

```
git tag -d tag_name
```

2. 再从远处删除

```
git push origin :refs/tags/tag_name
```

自定义git

让git显示颜色

```
git config --global color.ui true
```

忽略特殊文件

有些文件必须保存再工作区中，但是又不能提交，这时可以忽略这些文件

忽略文件原则：

- 操作系统自动生成的文件
- 编译生成的中间文件、可执行文件
- 带有敏感信息的文件

`.gitignore` 文件就是用于忽略特殊文件的,该文件的格式如下

```
# 这是注释
*.py #忽略所有拓展名为py的文件
Desktop.ini #忽略该名的文件
```

那么如果有个文件添加到了 `.gitignore` 中，怎么强制使其被git托管？

```
git add -f file.py
```

或者可能是 `.gitignore` 写的有问题，可以用 `git check-ignore -v file.py` 检查 `.gitignore` 文件是否对 `file.py` 文件的忽略策略有问题

最后，`.gitignore` 需要被git托管才可以

配置别名

- 假设为 `status` 这个命令配置别名为 `st`

```
`git config --global alias.st status`
```

`--global` :指全局参数，该配置在本台电脑的所有仓库下都有用。

- 为多个命令配置别名,如撤销修改中，`git reset HEAD test.txt` 是撤销暂存区的修改，将其重新放回工作区，是一个 `unstage` 操作，那么就可以为其配置一个 `unstage` 的别名

```
git config --global alias.unstage 'reset HEAD'
```

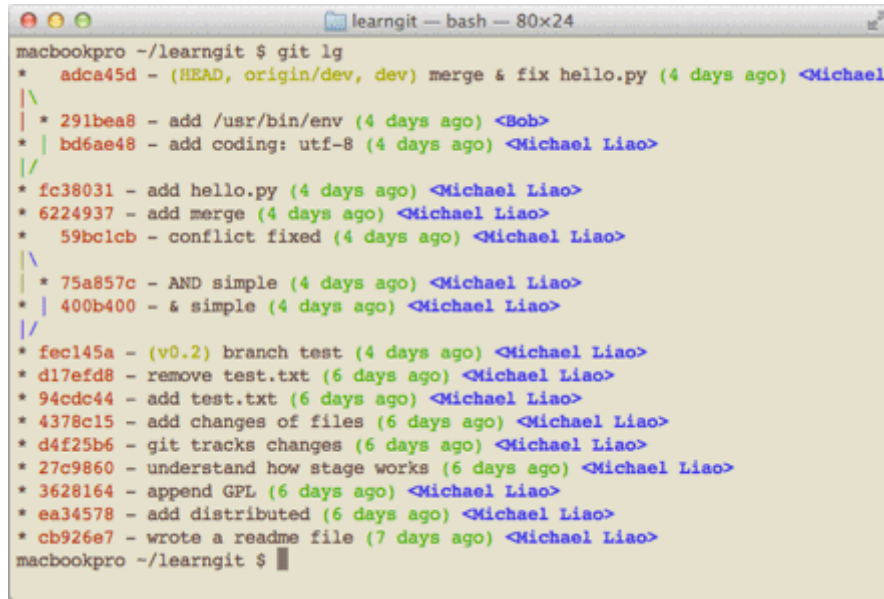
这样就可以使用 `git unstage test.txt` 撤销修改

- 配置 `git last` 让其显示最后一次提交信息

```
git config --global alias.last 'log -1'
```

- 还可以 `git config --global alias.lg "log --color --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold blue)<an>%Creset' --abbrev-commit"`

效果如下



```
macbookpro ~/learnig $ git lg
* adca45d - (HEAD, origin/dev, dev) merge & fix hello.py (4 days ago) <Michael
| \
| * 291bea8 - add /usr/bin/env (4 days ago) <Bob>
| * bd6ae48 - add coding: utf-8 (4 days ago) <Michael Liao>
| /
* fc38031 - add hello.py (4 days ago) <Michael Liao>
* 6224937 - add merge (4 days ago) <Michael Liao>
* 59bc1cb - conflict fixed (4 days ago) <Michael Liao>
| \
| * 75a857c - AND simple (4 days ago) <Michael Liao>
| * 400b400 - & simple (4 days ago) <Michael Liao>
| /
* fec145a - (v0.2) branch test (4 days ago) <Michael Liao>
* d17efd8 - remove test.txt (6 days ago) <Michael Liao>
* 94cdc44 - add test.txt (6 days ago) <Michael Liao>
* 4378c15 - add changes of files (6 days ago) <Michael Liao>
* d4f25b6 - git tracks changes (6 days ago) <Michael Liao>
* 27c9860 - understand how stage works (6 days ago) <Michael Liao>
* 3628164 - append GPL (6 days ago) <Michael Liao>
* ea34578 - add distributed (6 days ago) <Michael Liao>
* cb926e7 - wrote a readme file (7 days ago) <Michael Liao>
macbookpro ~/learnig $
```

那这些配置放在那里了呢？

每个仓库的Git配置都在.git/config文件中

当前用户的配置,即加了 `--global` 参数的配置放在了用户主目录下的一个隐藏文件 `.gitconfig` 中

管理公钥

[Gitis](#)

控制权限

[Gitolite](#)