

目录

Introduction	1.1
Git 介绍	1.2
资料收集整理	1.2.1
安装	1.3
Windows	1.3.1
CentOS 6.5	1.3.2
Ubuntu 16.04	1.3.3
配置 SSH	1.3.4
Git命令	1.4
branch/分支	1.4.1
创建空白分支	1.4.1.1
commit/提交	1.4.2
撤销提交	1.4.2.1
修改注释	1.4.2.2
merge/合并	1.4.3
更新github上fork的仓库	1.4.3.1
tag/标签	1.4.4
tag命令资料	1.4.4.1
Github	1.5
更新fork的仓库	1.5.1
Gitlab	1.6
安装配置	1.6.1
导入已有仓库	1.6.2
SourceTree	1.7
安装	1.7.1
设置	1.7.2
仓库	1.7.3
分支	1.7.4
查看提交信息	1.7.5
基本操作	1.7.6

Pull 时冲突	1.7.7
外部 Merge 工具	1.7.8
Merge 时冲突	1.7.9

安装**Git**

下面详细介绍如何在各个操作系统中安装git。

在windows上安装Git

下载

在 [git官网](#) 的下载页面找到windows版本

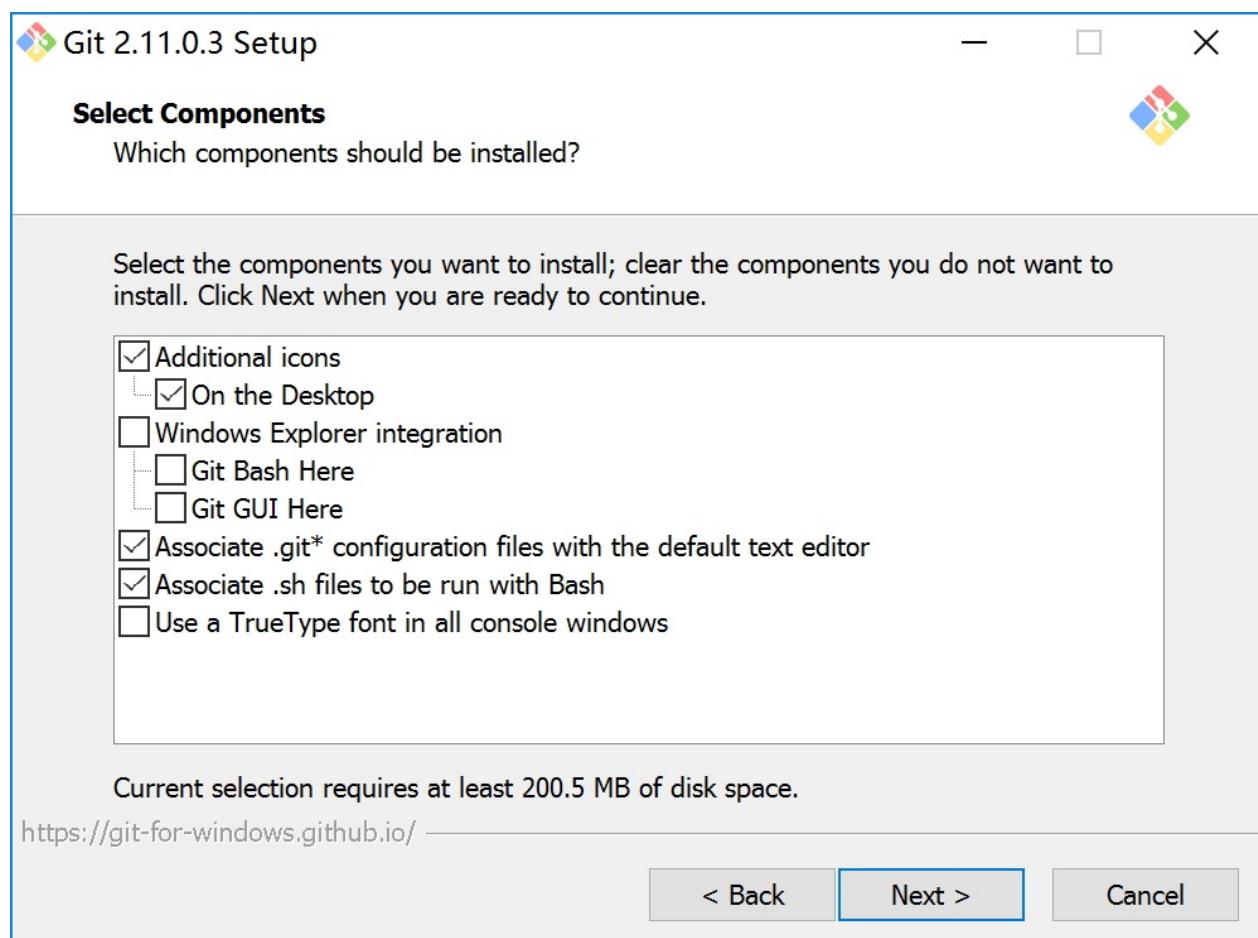
<https://git-scm.com/download/win>

下载得到安装包，如 ·Git-2.11.0.3-64-bit.exe·

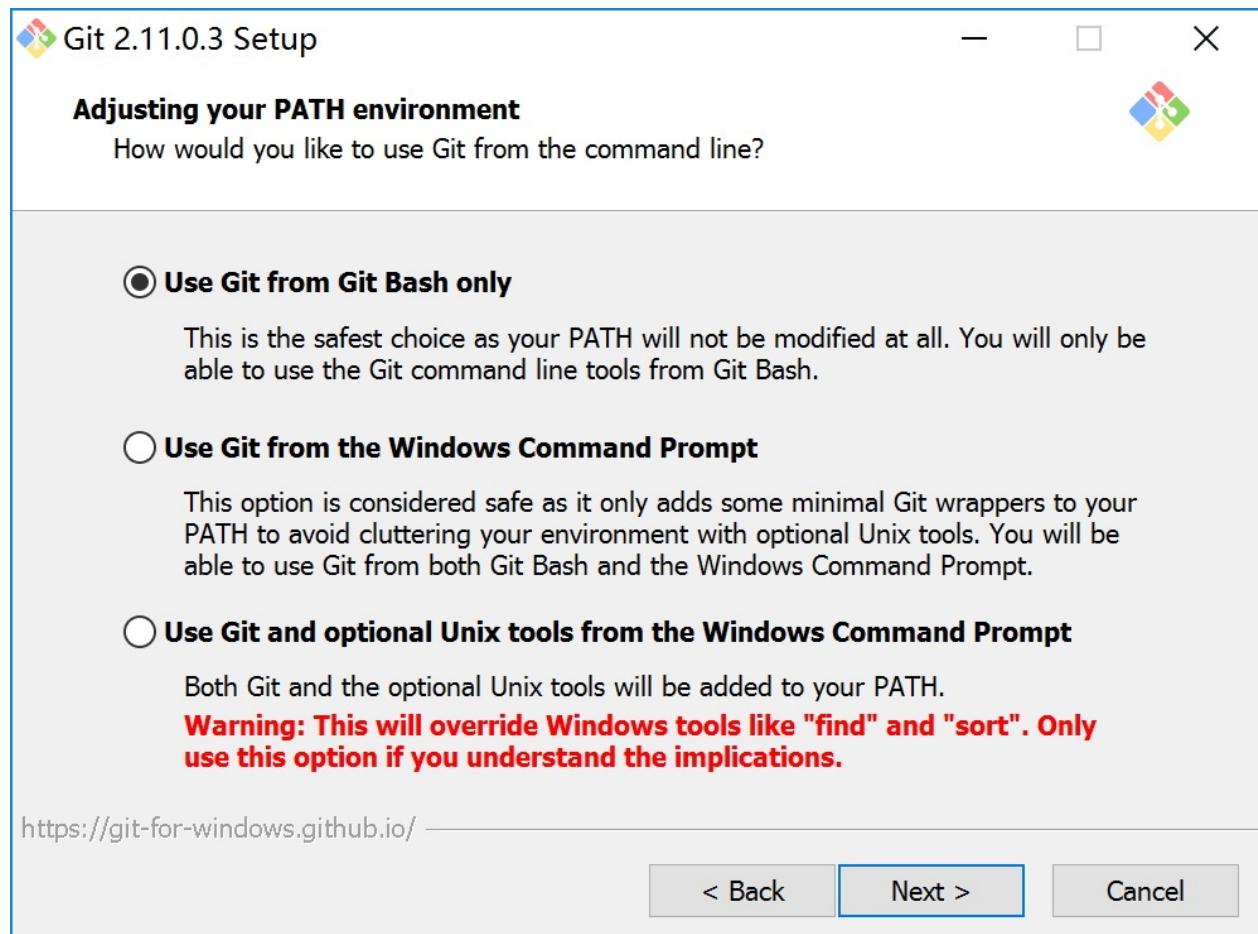
安装

安装过程基本按照默认设置，一路next就好了。

不过个人不喜欢引入太多的东西进入操作系统，所以个别设置会进行修改。



默认 "Git Bash Here" 和 "Git GUI Here" 是勾选的，我一般去掉。



默认是第二个，我一般选第一个，只在git bash里面用git。

Git 2.11.0.3 Setup

Configuring the line ending conversions

How should Git treat line endings in text files?

Checkout Windows-style, commit Unix-style line endings

Git will convert LF to CRLF when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform projects, this is the recommended setting on Windows ("core.autocrlf" is set to "true").

Checkout as-is, commit Unix-style line endings

Git will not perform any conversion when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform projects, this is the recommended setting on Unix ("core.autocrlf" is set to "input").

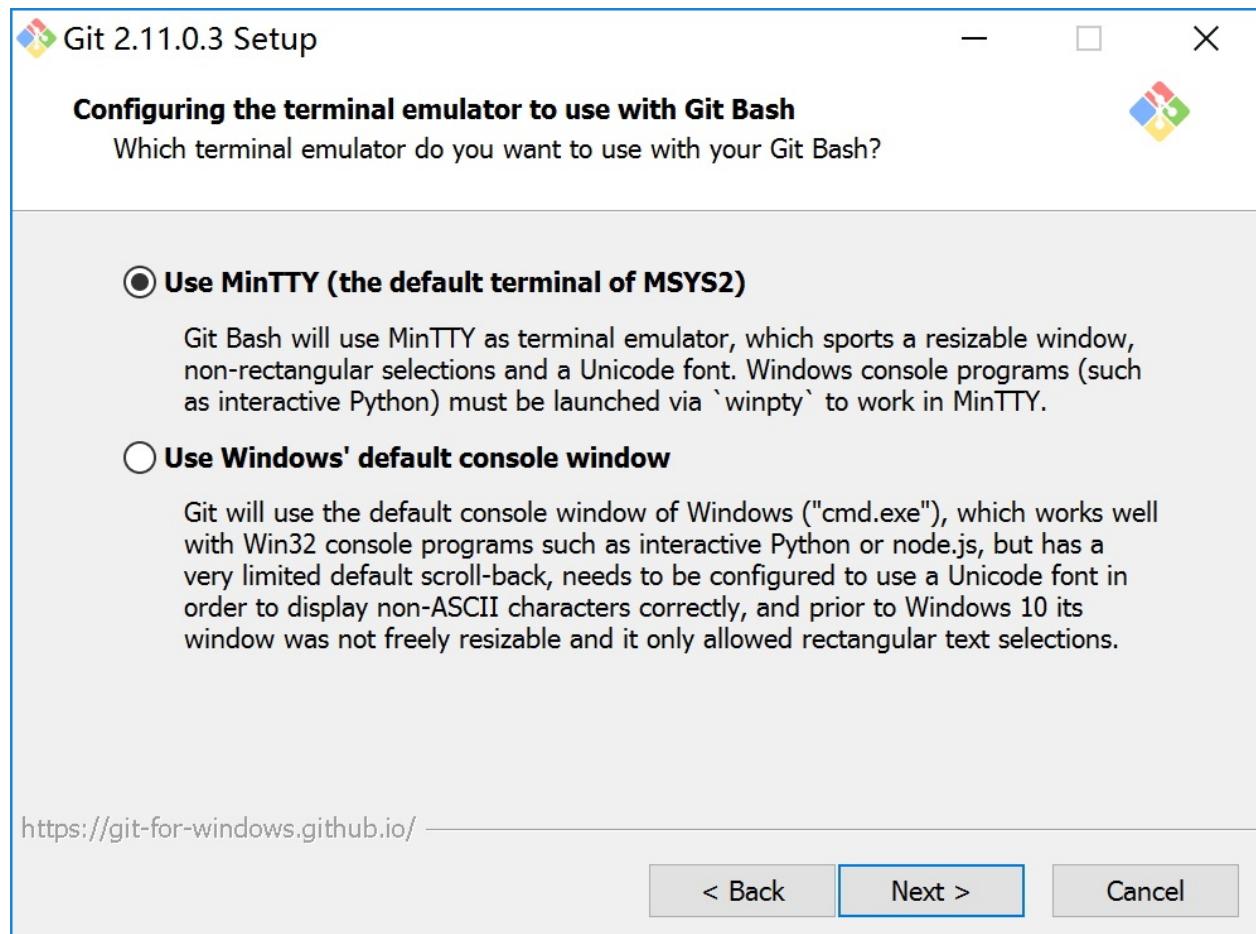
Checkout as-is, commit as-is

Git will not perform any conversions when checking out or committing text files. Choosing this option is not recommended for cross-platform projects ("core.autocrlf" is set to "false").

<https://git-for-windows.github.io/>

< Back Next > Cancel

务必选第一个。



terminal 建议还是用MinTTY，window的 cmd 实在不好用。

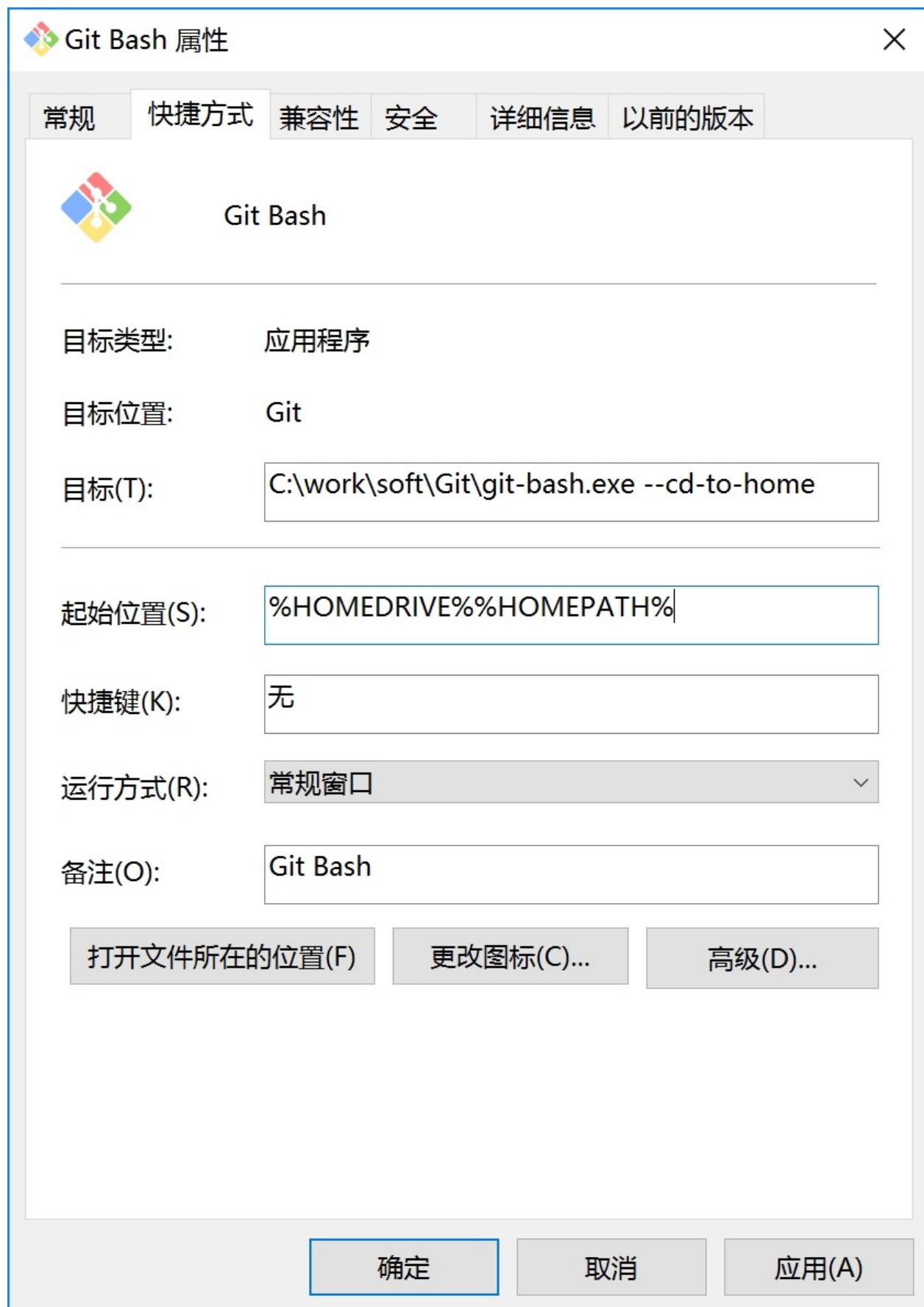
配置

修改启动后的默认路径

安装完成之后，桌面的 Git Bash 快捷方式打开之后，默认进入当前用户的home目录。

如果需要修改，比如我个人习惯直接进入代码存放路径如 C:\work\code，则需要修改快捷方式的属性。

编辑快捷方式的属性：

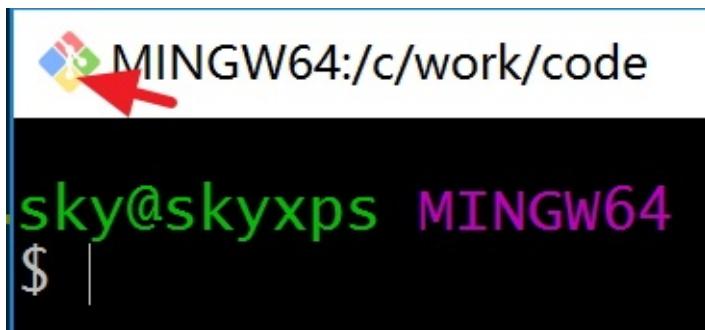


目标中删除 `--cd-to-home`，然后将起始位置从默认的 `%HOMEDRIVE%%HOMEPATH%` 修改为 `C:\work\code`。

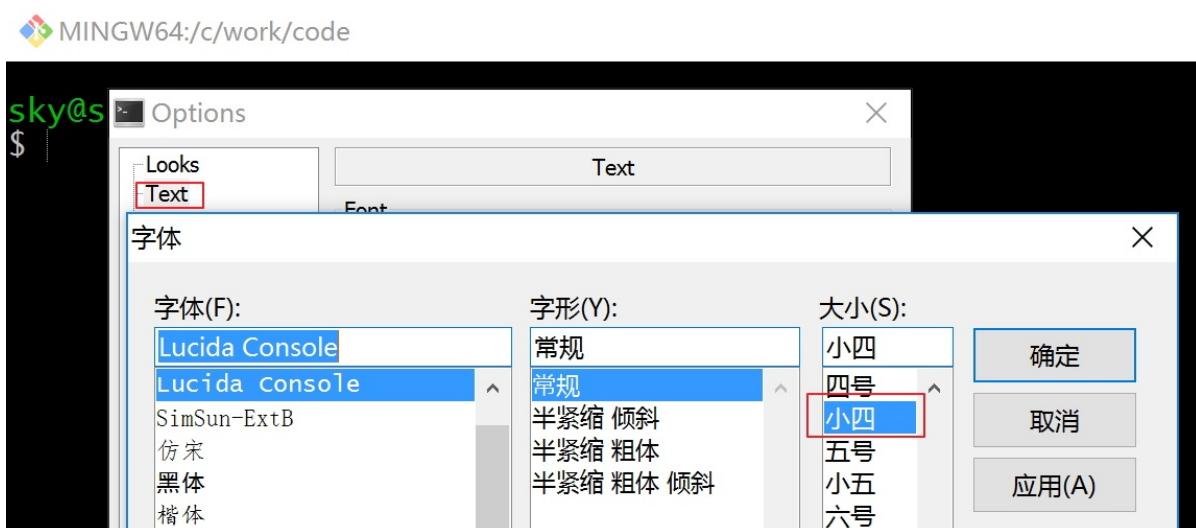
修改控制台字体

Git Bash 控制台默认的字体如果不合适（通常字体偏小），可以修改。

1. 点下图中红色箭头处，下拉菜单中选择"Options"



2. 在 "Text" 中可以选择字体和大小



CentOS 6.5 上安装git

前言

CentOS 6.5 上默认安装的git版本为 1.7.1, 版本太低, 报错如下:

```
using .gitcredentials to set credentials
[WARNING] Installed git version too old for credentials support
```

因此必须升级到新版本.

安装方式

yum 安装

依次执行以下命令:

```
rpm -i http://pkgs.repoforge.org/rpmforge-release/rpmforge-release-0.5.3-1.el6.rf.x86_
64.rpm
yum --enablerepo=rpmforge-extras update
yum --enablerepo=rpmforge-extras provides git
```

在列出的版本列表中选择最新的一个版本, 如:

```
git-1.7.12.4-1.el6.rfx.x86_64 : Git core and tools
Repo       : rpmforge-extras
Matched from:
```

执行安装命令:

```
yum --enablerepo=rpmforge-extras install git-1.7.12.4-1.el6.rfx.x86_64
```

安装结束之后, 通过"git version" 命令可以查看更新之后的版本信息:

```
git version 1.7.12.4
```

资料参考

- [CentOS 6.x 升级 Git](#): 主要参考这个文章，除了安装git那里命令有错-:-)

Ubuntu 16.04

安装

为了拿到最新版本的 git，增加仓库然后再安装：

```
sudo add-apt-repository ppa:git-core/ppa  
sudo apt-get update  
sudo apt-get install git
```

安装完成之后，验证版本：

```
git --version  
git version 2.15.0
```

此时 git 是在 /usr/bin/ 目录下：

```
root@sky2:~# which git  
/usr/bin/git  
root@sky2:~# ls -l /usr/bin/git  
-rwxr-xr-x 1 root root 1920512 Oct 4 03:50 /usr/bin/git
```

参考资料

- [Installing Latest version of git in ubuntu](#)

配置 ssh

为了方便日常使用，推荐采用 ssh 的方式来 clone git 仓库。

为此，需要创建 ssh key 并配置。

创建 ssh key

```
ssh-keygen -t rsa
```

一路回车确认，最后成功生成，创建的 ssh key 文件位于 `~/.ssh/id_rsa.pub`：

```
$ ls -l ~/.ssh
total 8
-rw----- 1 sky sky 1679 Oct 26 15:24 id_rsa
-rw-r--r-- 1 sky sky 396 Oct 26 15:24 id_rsa.pub
```

`id_rsa.pub` 是 ssh 的公钥，后面配置时要提交的公钥的内容就在这个文件中。

github 配置

登录之后，点红色箭头所指处，下拉菜单中点 "settings"，然后点 "SSH and GPG keys" -> "New SSH key"，最后将 `id_rsa.pub` 文件的内容复制过来提交：

The screenshot shows the GitHub user profile page. On the left, there's a sidebar with links: Personal settings, Profile, Account, Emails, Notifications, Billing, SSH and GPG keys (with a red circle '2'), Security, Blocked users, and Repositories. The main content area is titled 'SSH keys'. It says: 'This is a list of SSH keys associated with your account. Remove any keys that you do not recognize.' Below this, there are two entries:

- sky@xps** (key icon) **Fingerprint:** c3:f7:80:a4:54:93:d3:f2:3b:f3:85:b6:21:a9:bf:1a **SSH** **Delete** **Added on 17 Oct 2016 — Last used within the last day**
- sky@basiccloud** (key icon) **Fingerprint:** 13:bf:e8:c6:4f:4b:55:60:86:c7:40:a2:5d:99:94:dc **SSH** **Delete** **Added on 26 Oct 2016 — Never used**

At the bottom of the 'SSH keys' section, there's a note: 'Check out our guide to generating SSH keys or troubleshoot common SSH Problems.'

配置完成之后，尝试以ssh方式执行git clone 命令：

```
git clone git@github.com:basiccloud/basiccloud-site.git
```

gitlab 配置

登录gitlab之后，在当前用户的 "Profile Settings" -> "SSH Keys" 中，提交id_rsa.pub 中公钥的内容。

操作

branch / 分支命令

git中创建新的空白分支

在偶尔的情况下，可能会想要保留那些与你的代码没有共同祖先的分支。例如在这些分支上保留生成的文档或者其他一些东西。

如果需要创建一个不使用当前代码库作为父提交的分支，可以用如下的方法创建一个空分支。

方法1

执行以下git命令：

```
git symbolic-ref HEAD refs/heads/newbranch  
rm .git/index  
git clean -fdx  
<do work>  
git add your files  
git commit -m 'Initial commit'
```

方法2

这里以github的操作为例，下面试图创建一个名为gh-pages的空分支：

```
$ cd repo  
  
$ git checkout --orphan gh-pages  
# 创建一个orphan的分支，这个分支是独立的  
Switched to a new branch 'gh-pages'  
  
$ git rm -rf .  
# 删除原来代码树下的所有文件  
rm .....
```

添加内容并push

注意这个时候用git branch命令是看不见当前分支的名字的，除非进行了第一次commit。

下面我们开始添加一些代码文件，例如这里新增了一个index.html:

```
$ echo \"My GitHub Page\" > index.html  
$ git add .  
$ git commit -a -m \"First pages commit\"  
$ git push origin gh-pages
```

在commit操作之后，你就可以用git branch命令看到新分支的名字了，然后push到远程仓库。

提交 / **commit**

撤销commit

背景

- 代码修改后已经执行了commit和push
- 希望撤销这次commit，回到到这个commit之前的状态：包括本地和远程仓库

操作

找出提交历史记录

先 `git log` 找出提交的历史记录

比如下面有四个commit记录：

```
$ git log
commit 87dabb290ae1a4e620512b7cd81d2161747c6ec9
Author: Sky Ao <aoxiaojian@gmail.com>
Date:   Thu Jul 21 16:36:26 2016 +0800

    add client builder and wrap native api; setup integration test; add first unit test
    case and integration test case

commit 6899dd19dbe58da6ae65fd157a791151967c16b2
Author: Sky Ao <aoxiaojian@gmail.com>
Date:   Thu Jul 21 15:17:00 2016 +0800

    rollback package name to etcdserverpb, otherwise etcd server will reject the request

commit 18d034b3a1e20e81ece3d2f6ba9919e8bdb3dfd4
Author: Sky Ao <aoxiaojian@gmail.com>
Date:   Thu Jul 21 14:37:19 2016 +0800

    change java version to 1.7

commit 4beef3ce6557a41e657c2ee4e19c6156eabd759b
Author: Sky Ao <aoxiaojian@gmail.com>
Date:   Wed Jul 20 18:40:18 2016 +0800
```

现在需要撤销最新的这一次 `87dabb290ae1a4e620512b7cd81d2161747c6ec9` 提交，回退到它的上一次 `6899dd19dbe58da6ae65fd157a791151967c16b2`。

执行 **reset** 命令

```
git reset --hard 6899dd19dbe58da6ae65fd157a791151967c16b2
HEAD 现在位于 6899dd1 rollback package name to etcdserverpb, otherwise etcd server will
reject the request
```

注意 commitid 是要撤销的commit的前一次commit的id，也就是说 **reset** 命令是将提交重置到要撤销的前一次。

reset命令执行完成后，本地仓库就重置，相当于撤销了 6899dd1 之前的所有commit。

执行强制的 **push** 命令

执行 **push** 命令，注意是需要增加 **--force** 来强制推送：

```
$ git push origin HEAD --force
Total 0 (delta 0), reused 0 (delta 0)
To git@github.com:skyao/jetcd.git
+ 87dabb2...6899dd1 HEAD -> master (forced update)
```

push 命令执行完成，远程仓库也就重置了。此时从远程仓库上看，在这次 6899dd19dbe58da6ae65fd157a791151967c16b2 提交之后的所有commit已经消失，相当于 git 仓库回滚到这个提交了。

处理受保护分支

如果当前分支是 **protected** 的受保护分支，则 git 服务器会拒绝强制推送，报错如下：

```
$ git push origin HEAD --force
Total 0 (delta 0), reused 0 (delta 0)
remote: GitLab: You are not allowed to force push code to a protected branch on this p
roject.
To basiccloud.net:foundation/foundation-etcd.git
 ! [remote rejected] HEAD -> master (pre-receive hook declined)
error: 无法推送一些引用到 'git@basiccloud.net:foundation/foundation-etcd.git'
```

此时，需要将当前分支的 **Protected** 属性暂时去掉。

对于 gitlab，可以进入当前仓库的设置中的 "Protected branches" 一项，将当前 branch 的 **protected** 临时去掉，等这次 **push** 完成，再重新设置回 **protected** 。

参考资料

- git 删除错误提交的commit

修改注释

已经 **commit** 但还没有 **push**

TBD

已经 **push**

对于已经提交并已经 push 到远程仓库中的需要通过 `git rebase` 才能完成。

首先要 `git rebase` 到需要修改的那个 commit 的前1个 commit。假设 commit id 是 `32e0a87f`，运行下面的 `git rebase` 命令：

```
git rebase -i 32e0a87f
```

在 git bash 中运行上面的命令后，会弹出编辑框，在编辑框中会分行依次显示以 `pick` 开头的这个 commit 之后的所有 commit message。

将需要修改的commit message之前的"pick"改为"reword"，点击保存按钮，并关闭编辑框，这时会执行rebase操作。

Rebasing (1/3)

接着会再次弹出编辑框，这次编辑框中只有之前改为"reword"的那个commit message，此时修改commit message的内容，点击保存按钮并关闭编辑框，会继续执行rebase操作。

如果操作成功，会出现如下的提示：

```
[detached HEAD aa3b52c] Add return url
 2 files changed, 1 insertion(+), 3 deletions(-)
 Successfully rebased and updated refs/heads/oss.
```

这样就完成了git commit message的修改，然后强制push一下就搞定了。

```
git push --force
```

参考资料

- <http://www.cnblogs.com/dudu/p/4705247.html>

- <https://help.github.com/articles/changing-a-commit-message/>
- 快速批量修改Git提交注释方法

merge / 合并

更新github上fork的仓库

背景

- 在github上fork了某项目
- 原仓库有新的改动
- 想将原仓库的改动更新到自己fork的仓库

操作过程

以netty为例：

- 源地址：git@github.com:netty/netty.git
- 我fork的：git@github.com:skyao/netty.git

按照下面的步骤：

1. 为本地仓库增加一个remote, 命名为"upstream":

```
git remote add upstream git://github.com/netty/netty.git
```

也有人推荐下面的多了--track参数的方式：

```
git remote add --track master upstream git://github.com/netty/netty.git
```

2. fetch 这个upstream远程的所有分支到remote-tracking分支, 例如upstream/master

```
git fetch upstream
```

3. 确认当前分支是master分支, 如果不是checkout到master分支

```
git branch  
git checkout master
```

4. 同步upstream的修改到本地, 可以选择rebase或者merge

```
git rebase upstream/master  
git merge upstream/master
```

注: 推荐用merge.

5. 将更新之后的版本推送到自己fork的仓库

```
git push -f origin master
```

参考资料

- Pull new updates from original Github repository into forked Github repository
- How to update a GitHub forked repository?
- HOW TO GITHUB: FORK, BRANCH, TRACK, SQUASH AND PULL REQUEST

tag / 标签命令

tag命令资料

记录git中tag命令的用法。

tag命令用法

列出已有tag

```
git tag  
git tag -l 'v1.4.2.*'
```

添加tag

添加轻量级标签：

```
git tag -a v1.4
```

添加含附注类型的标签：

```
git tag -a v1.4 -m 'my version 1.4'
```

推送到远程：

```
git push origin v1.4
```

或者--tags推送全部：

```
git push --tags
```

删除tag

删除本地tag：

```
git tag -d v1.0.0
```

删除远程tag：

```
git push origin :refs/tags/v1.0.0
```

资料

git官方资料

中英文两个版本：

- [2.6 Git Basics - Tagging](#)
- [2.6 Git 基础 - 打标签](#)

Github

更新fork的仓库

在github上fork了一些仓库，需要更新到最新版本，记录方法和过程。

前言

需要将自己fork的github项目更新，google了一下方法，记录下来。

参考资料：

- [Pull new updates from original Github repository into forked Github repository](#)
- [How to update a GitHub forked repository?](#)
- [HOW TO GITHUB: FORK, BRANCH, TRACK, SQUASH AND PULL REQUEST](#)

更新过程

按照下面的步骤：

git@github.com:netty/netty.git

1. 为本地仓库增加一个remote，命名为"upstream":

```
git remote add upstream git://github.com/netty/netty.git
```

也有人推荐下面的多了--track参数的方式：

```
git remote add --track master upstream git://github.com/netty/netty.git
```

2. fetch 这个upstream远程的所有分支到remote-tracking分支，例如upstream/master

```
git fetch upstream
```

3. 确认当前分支是master分支，如果不是checkout到master分支

```
git branch git checkout master
```

4. 同步upstream的修改到本地，可以选择rebase或者merge

```
git rebase upstream/master git merge upstream/master
```

注：还是推荐用merge。

5. 将更新之后的版本推送到自己fork的仓库

```
| git push -f origin master
```

Gitlab



- <https://about.gitlab.com/>

gitlab 是目前最为流行的开源git管理工具，强烈推荐。

Gitlab 安装

安装

安装方式参考gitlab官方资料，打开 <https://about.gitlab.com/downloads/> 页面，"select operating system"选对应的版本，然后按照指示执行安装。

- ubuntu 14.04

```
sudo apt-get install openssh-server
sudo apt-get install postfix
wget https://downloads-packages.s3.amazonaws.com/ubuntu-14.04/gitlab_7.7.2-omnibus.5.4.2.ci-1_amd64.deb
sudo dpkg -i gitlab_7.7.2-omnibus.5.4.2.ci-1_amd64.deb
```

- ubuntu 16.04

```
sudo apt-get install curl openssh-server ca-certificates postfix
curl -sS https://packages.gitlab.com/install/repositories/gitlab/gitlab-ce/script.deb.sh | sudo bash
sudo apt-get install gitlab-ce
```

中间安装 postfix 时，选择 "Internet Site"，然后 system mail name 填写域名如 "basiccloud.net"。

安装过程结束之后再执行命令：

```
sudo gitlab-ctl reconfigure
```

之后就可以通过浏览器访问了，默认是用80端口。

配置

修改端口

安装完成之后默认是用80端口，如果需要用其他端口，需要修改一下配置文件。

```
sudo vi /etc/gitlab/gitlab.rb
```

修改`external_url`，直接增加端口号即可，比如我这里用8800端口：

```
external_url 'http://skyserver:8800'
```

也可以在这里修改`host`为真实的公网`host`,比如"<http://basiccloud.net:8800>"

修改后再次执行"`sudo gitlab-ctl reconfigure`"以便配置修改生效。

注意：除了这个端口外，还有一个`unicorn`用的端口，默认是8080，如果8080端口被其他程序占用。那么`unicorn`就会无法启动，显示为502错误，“GitLab is not responding”。

这种情况下修改`unicorn`的配置(默认下面两个配置是被注释的，需要自己打开)：

```
unicorn['listen'] = '127.0.0.1'  
unicorn['port'] = 8811
```

完成后通过浏览器访问(<http://basiccloud.net:8800>).

补充: 在ubuntu14.04服务器上,发现按照上面的修改并执行"`sudo gitlab-ctl reconfigure`",会发现默认的8080和新修改的8801两个端口都会同时被gitlab占用. 最后只有重启ubuntu服务器才能释放出8080端口,原因不明.

账号设置

默认管理员密码如下：

```
Username: root  
Password: 5iveL!fe
```

`root`账户第一次登录时会要求修改密码，为了安全我们在管理页面可以新建一个普通用户，注意新建用户过程中不能设置密码，在建立成功之后可以edit这个账号然后这里可以设置密码。

关闭注册功能

默认注册功能是开启的，对于个人的gitlab, 没有对外公布的必要(有就直接上github了)，因此需要考虑关闭注册功能。

用管理员账号登录之后，进入"Admin area", 点"settings", 取消"Signup enabled".

配置反向代理

按照上述安装方式，gitlab 服务于下面的地址：

<http://basiccloud.net:8800>

由于携带端口号，会造成输入和记忆的麻烦，因此推荐在前面加一个 nginx 做反向代理，将地址转为 <http://git.basiccloud.net> 这样的比较友好的形式。

具体操作请见 nginx 学习笔记 中的反向代理的使用：

<https://skyao.gitbooks.io/leaning-nginx/>

导入已有仓库

记录如何将现有的git repository导入到gitlab中.

前言

在使用 gitlab 之前, 我使用 gitolite 来保存自己私有的代码. 后来发现 gitlab 的功能远比 gitolite 要强大, 使用上也方便很多. 因此考虑放弃 gitolite, 而原有在 gitolite 中的代码就需要迁移过去, 同时希望能保留原有的 commit/tag/branch 等信息.

批量导入

发现 gitlab 对此有特别的支持, 而且支持多个仓库一起导入, 非常方便处理大批仓库的迁移.

参考官方文档 : [Import bare repositories into your GitLab instance](#)

假设我们有三个项目,a.git/b.git/c.git在gitolite中, 我们准备将他们导入gitlab, 放在名为backup的group中。则导入操作步骤如下:

1. 打包原有的bare reposotory

注意一定要是**bare reposotory**。

可以直接到git仓库的原始存储路径下打包, 如 :

```
cd /home/git/repositories/  
tar cvf code.tar a.git b.git c.git
```

也可以通过带 --bare 参数的git clone命令来获取 :

```
git clone --bare xx/a.git
```

2. 在gitlab中为将要导入的项目做准备

在gitlab的管理界面上添加一个名为backup的group.

3. 将打包后的文件传到目标机器上

4. 准备导入的目录结构

```
# import目录下存放所有要导入的仓库，放在哪里无所谓
mkdir /home/sky/import/
# 建立子目录，对应要导入的group名
mkdir /home/sky/import/backup

# 将要导入的git仓库复制到子目录下
cp -R a.git b.git c.git /home/sky/import/backup

# 将整个import目录的owner和group都修改为git
sudo chown -R git:git /home/sky/import/
```

5. 执行导入命令

```
# 注意这里的路径只到import路径，不要写子路径，子路径是用来制定group的
sudo gitlab-rake gitlab:import:repos['/home/sky/import/']
```

命令执行输出类似如下：

```
Processing /home/sky/import/backup/a.git
* Using namespace: backup
* Created a (backup/a-service)
```

成功之后就可以在gitlab的页面上看到新导入的git项目。

单个导入

先clone要迁移的仓库，clone的时候用--mirror参数来clone它的bare repository，然后添加remote：

```
git clone --mirror git@192.121.166.180:root/ut-cidemo.git
cd ut-cidemo.git
git remote add gitlab git@basiccloud.net:lagency/ut-cidemo.git
git push -f --tags gitlab refs/heads/*:refs/heads/*
```

这种方式适合导入单个仓库。

SourceTree



SourceTree

sourcetree 是一个免费的git和hg客户端，支持 Mac 和 windows。

A free visual Git and Hg client for Mac and Windows

The screenshot shows the SourceTree application window. The top menu bar includes 'sourcetree-website (Git)', 'Commit', 'Pull', 'Push', 'Branch', 'Merge', 'Shelve', 'Show in Finder', 'Terminal', and 'Settings'. The left sidebar has sections for 'WORKSPACE' (File status, History, Search), 'BRANCHES', 'BOOKMARKS', 'TAGS', 'REMOTES', 'SHELVED', and 'SUBREPOSITORIES'. The main area displays a 'Graph' view of the commit history, showing a vertical timeline with colored dots representing commits. A list of commits is shown below, with columns for Commit ID, Author, Description, and Date.

Commit	Author	Description	Date
b7358c7	Rahul Chhab...	[b] master [b] origin/master [b] origin/HEAD Removing ol...	Mar 3, 2016, 11:...
bdb8bef	Rahul Chhab...	Merged in update-google-verification (pull request #14)	Feb 18, 2016, 1:3...
dfe975d	Tyler Tadej...	[b] origin/update-google-verification Update google verificati...	Feb 11, 2016, 2:2...
3bc3290	Tyler Tadej...	Replace outdated Atlassian logo in footer with base-64 en...	Feb 11, 2016, 2:1...
dba47f9	Tyler Tadej...	Add gitignore	Feb 11, 2016, 1:3...
ff67b45	Mike Minns...	Updated Mac min-spec to 10.10	Feb 15, 2016, 11:...
72d32a8	Michael Min...	Merged in hero_images (pull request #13)	Feb 15, 2016, 10:...
246c4ff	Joel Unger...	[b] origin/hero_images [b] hero_images Used TinyPNG to c...	Feb 11, 2016, 3:3...
9d9438c	Joel Unger...	Replacing hero images with new version of SourceTree	Feb 9, 2016, 2:59...
ce75b63	Michael Min...	Merged in bug/date-https (pull request #12)	Feb 15, 2016, 10:...
85367bb	Patrick Tho...	[b] origin/bug/date-https fixed date and https errors	Jan 7, 2016, 12:2...
4f9b557	Joel Unger...	New Favicon	Feb 8, 2016, 3:55...
384e6d5	Rahul Chhab...	[b] origin/search-console-access search console google ver...	Feb 3, 2016, 2:09...
6fa47a9	Mike Minns...	updated to move supported version to OSX 10.9+	Dec 15, 2015, 2:0...
8dd87bb	Mike Minns...	remove extra , when a line is skipped due to empty server	Nov 23, 2015, 2:2...
faa195e	Mike Minns...	Skip records with empty server/user id as gas rejects them	Nov 23, 2015, 2:1...
0cdfe96	Mike Minns...	corrected paths after merge	Nov 23, 2015, 2:0...
051ab1b	Mike Minns...	corrected column counting	Nov 23, 2015, 1:5...
a723bc2	Mike Minns...	Merge branch 'au2gex'	Nov 23, 2015, 1:5...
65fd580	Mike Minns...	deal with invalid instanceids	Nov 23, 2015, 1:5...
500a892	Michael Min...	Merged in au2gex (pull request #11)	Nov 23, 2015, 1:0...

口号：在一个非常简单的应用程序中利用 Git 和 Hg 的力量

Harness the power of Git and Hg in a beautifully simple application

SourceTree 简化了如何与 Git 和 Mercurial 存储库交互的方式，以便您可以专注于编码。通过 SourceTree 的简单界面可视化和管理您的存储库。

- 对初级用户入门简单：对命令行说再见 - 简化您的团队的分布式版本控制，并迅速使每个人都能够快速前行
- 对专家功能强大：完美，让高级用户更加高效。查看分支机构之间的更改，stash/隐藏，cherry-pick等。

docs/themes/sourcetree/templates/base.html

Hunk 1 : Lines 1-7

```

1 1  <!DOCTYPE html>
2 2  <html>
3 3  <head>
4 + <title>SourceTree Help</title>
5 5  <link rel="stylesheet" href="{{ SITEURL }}/{{ THEME_STATIC_DIR }}/css/
6 6  </head>
7 7  <body>
```

Hunk 2 : Lines 15-24

```

14 15 <footer>
15 16   <a href="http://blog.sourcetreeapp.com/">Blog</a> | <a href="http://w
16 17
17 -   Copyright © 2014
18 -   <a href="http://www.atlassian.com/"><img href="{{ SITEURL }}/{{ THEME
18 + <div id="footer-right">
19 +   Copyright © 2014
```

- 可视化您的代码：眼见为实。一次点击即可获得任何分支或提交的信息。
- 桌面上的Git和Hg：全功能的GUI，即开即用，提供高效，一致的开发流程。

Sorted by path ▾

index.php

+ images/hero_mac_all.png

+ images/hero_mac_all@2x.png

+ images/hero_mac_bookmarks.png

+ images/hero_mac_bookmarks@2x.png

+ images/hero_mac_commit.png

 Merged in hero_images (pull request #13)

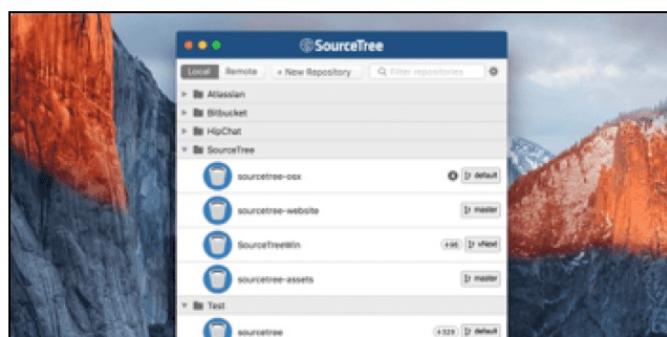
Replacing hero images with new version of SourceTree

Commit: 72d32a8164178ca9...
Parents: [ce75b635af](#), [246c4f...](#)
Author: Michael Minns [Atla...]
Date: February 15, 2016 ...

+ images/hero_mac_bookmarks@2x.png

New binary file After

After



安装

下载

在 sourcetree 网站下载对应的版本(windows或者mac)

<https://www.sourcetreeapp.com/>

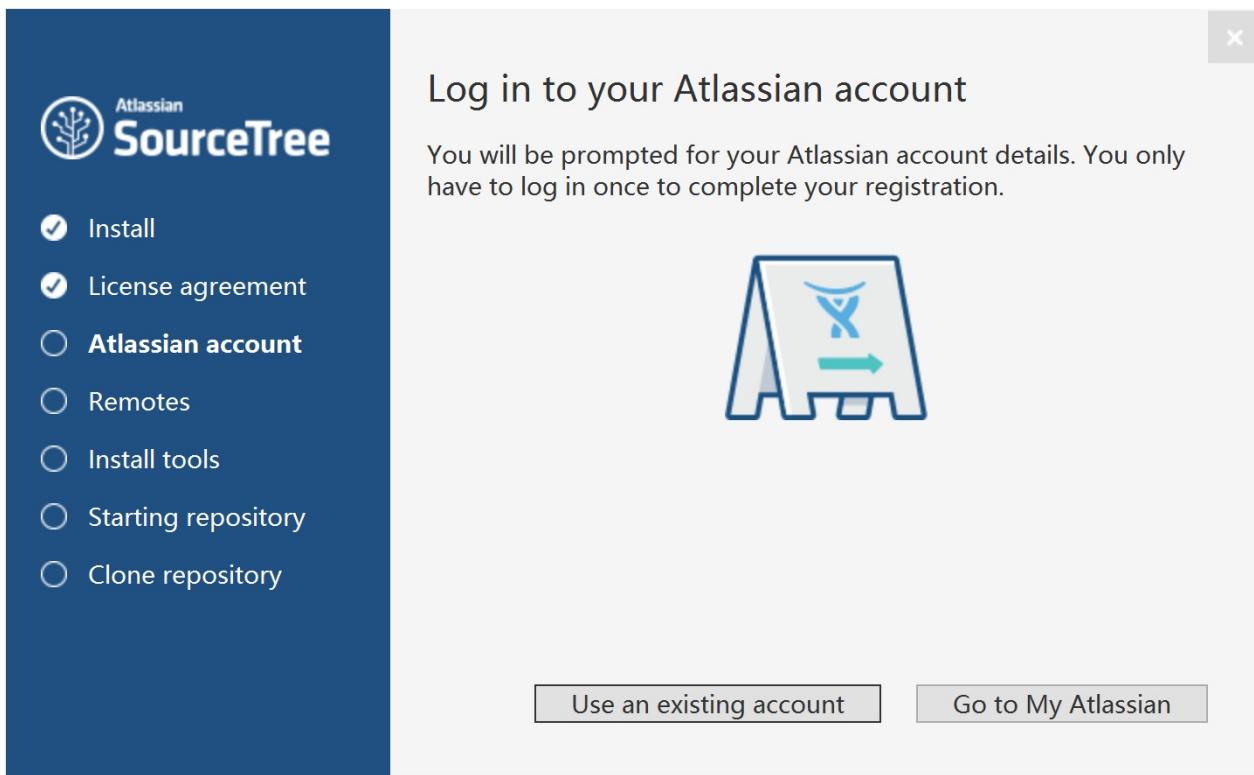
安装

打开下载的文件如 SourceTreeSetup-1.10.23.1.exe 开始安装.

步骤1：统一协议



步骤2：使用 Atlassian 账号登录



要求必须使用 Atlassian 账号登录才能继续，如果有账号的可以点 "Use an existing account" 直接登录，如果没有账号可以点 "Go to My Atlassian" 去注册，完成后再回来登录。

特别提醒：Atlassian 的网站采用了一种名为"人机身份验证"的程序来保护自己，避免被攻击。这个程序采用的是 Google ReCAPTCHA，因此对于大陆的同学可能因为被墙而无法使用，报错如下：



Having trouble logging in?

Send a recovery email to

aoxiaojian@hotmail.com

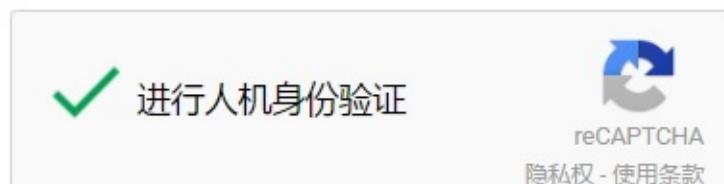
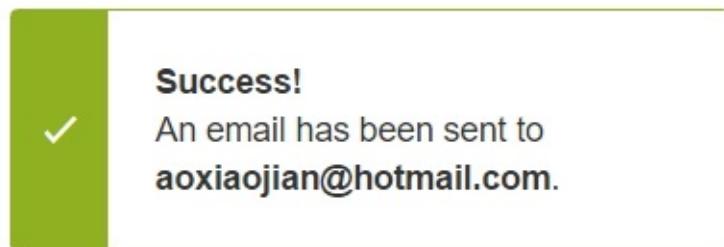
Google ReCAPTCHA failed to load

Send

翻墙之后就OK了，如图所示：

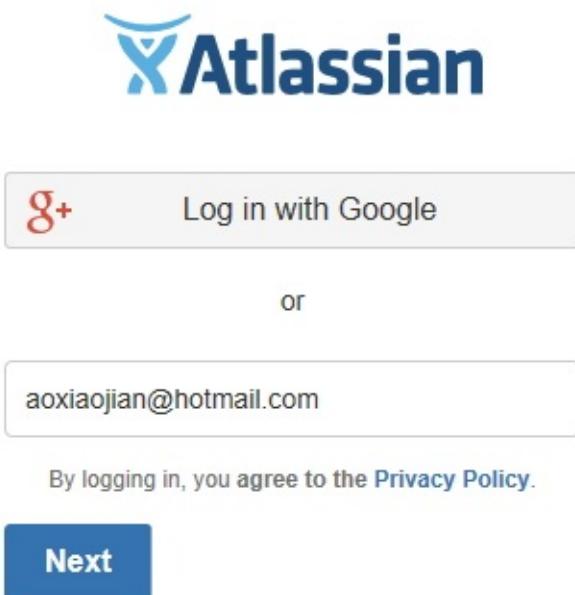


Having trouble logging in?



Resend

继续，点"Use an existing account" 开始登录，弹出窗口：



输入账号密码，登录成功后跳转到下面的窗口，就可以继续了：

Registration Complete!



You are logged in as **aoxiaojian@gmail.com**

继续

特别注意：如果是注册很多年的账号，会出现一个很奇怪的问题，输入账号密码登录之后，会停留在这个界面而不进行跳转，导致无法继续安装。经过检查是因为我这个账号很多年前就注册使用过，里面的 sourcetree 服务已经过期（2013年就过期）。

The screenshot shows the Atlassian account management interface. On the left sidebar, there are sections for GENERAL (Account settings, Change email address, Services), SECURITY (Change password), and BILLING (Billing details). The main area is titled "Account settings" and contains fields for Email (aoxiaojian@hotmail.com), Full name (Sky Ao), Job title, Organization (BasicCloud), and Timezone. A "Save" button is at the bottom. Below the form is a placeholder for an avatar with a "Change avatar" link.

新账号不会有这个问题，所以我被逼重新注册了另外一个新账号。

步骤3：创建账号

The screenshot shows the SourceTree "Connect an Account" setup screen. On the left, a sidebar lists steps: Install (checked), License agreement (checked), Atlassian account (checked), Remotes (radio button), Install tools (radio button), Starting repository (radio button), and Clone repository (radio button). The main area is titled "Connect an Account" and instructs users to connect to a remote server to clone existing repositories. It offers options for Bitbucket, Bitbucket服务器 (Bitbucket Server), and GitHub. Below these are fields for "托管 URL" (https://github.com/) and "验证" (OAuth). At the bottom are "跳过初始设置" (Skip initial setup) and "继续" (Continue) buttons.

如果不使用 BitBucket，可以点“跳过初始设置”。

步骤4：加载SSH密钥



如果有现成的，可以载入，没有就先跳过。

步骤5：设置git

SourceTree : 未找到 Git

i 我们未能检测到已经安装的 Git。有以下3种解决方式：

- 下载一个只被 SourceTree 使用的内嵌版 Git。
- 选择你的系统中 Git 的位置（如果没能自动找到）
- 重新检查（您刚刚已自行下载并安装完整版本的 Git）
- 我不想使用 Git

点“选择你的系统中git的位置”，然后选中你git安装路径中的git.exe,一般是 ·yourGit\bin\git.exe·

步骤6：设置Mercurial

SourceTree: 未找到 Mercurial



没有检测到已安装的 Mercurial。可以通过以下3种方式解决此问题：

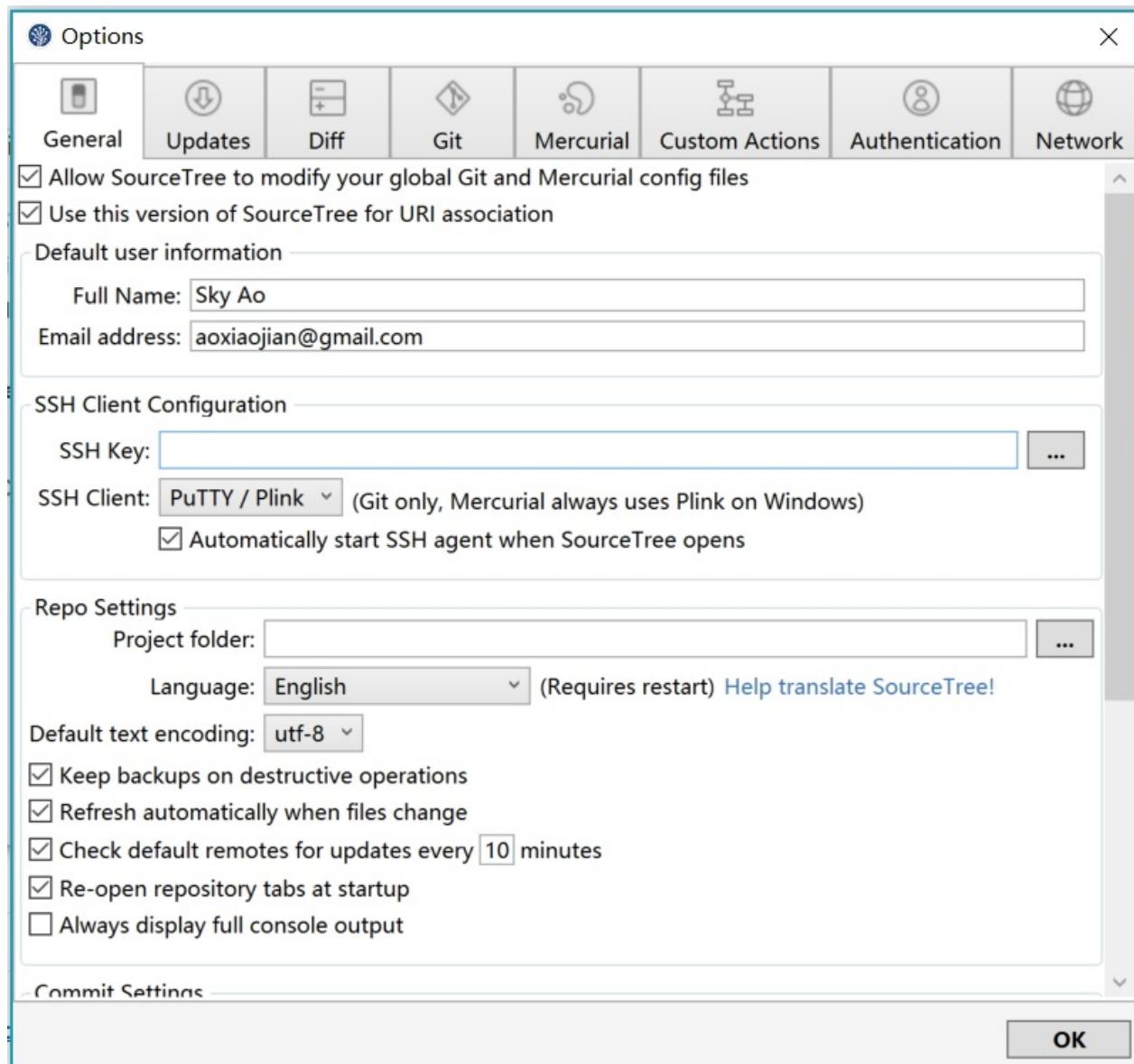
- 下载一个只被 SourceTree 使用的内嵌版 Mercurial。
- 选择您的系统中 Mercurial 的位置（如果没能自动找到）
- 再次检查（如果你刚刚自行下载并安装了一个完整版本的 Mercurial）
- 我不想使用 Mercurial

点"我不想使用Mercurial"，跳过。

设置

打开设置

在菜单中打开 "Tools" -> "Options" :



设置用户信息

设置默认用户信息

用户信息在每次提交/commit时都会附带上，以此来标志当前提交是谁提交的。

用户信息可以有两个层次：

- 仓库级别的用户信息：作用域仅仅限于当前仓库(repository)，不同仓库可以设置不同的信息。仓库级别的用户信息可以覆盖全局用户信息，如果仓库级别没有设置则取全局信息，如果都没有则提交时git会强制要求填写。
- 全局(global)用户信息：也被成为默认用户信息(default user information)

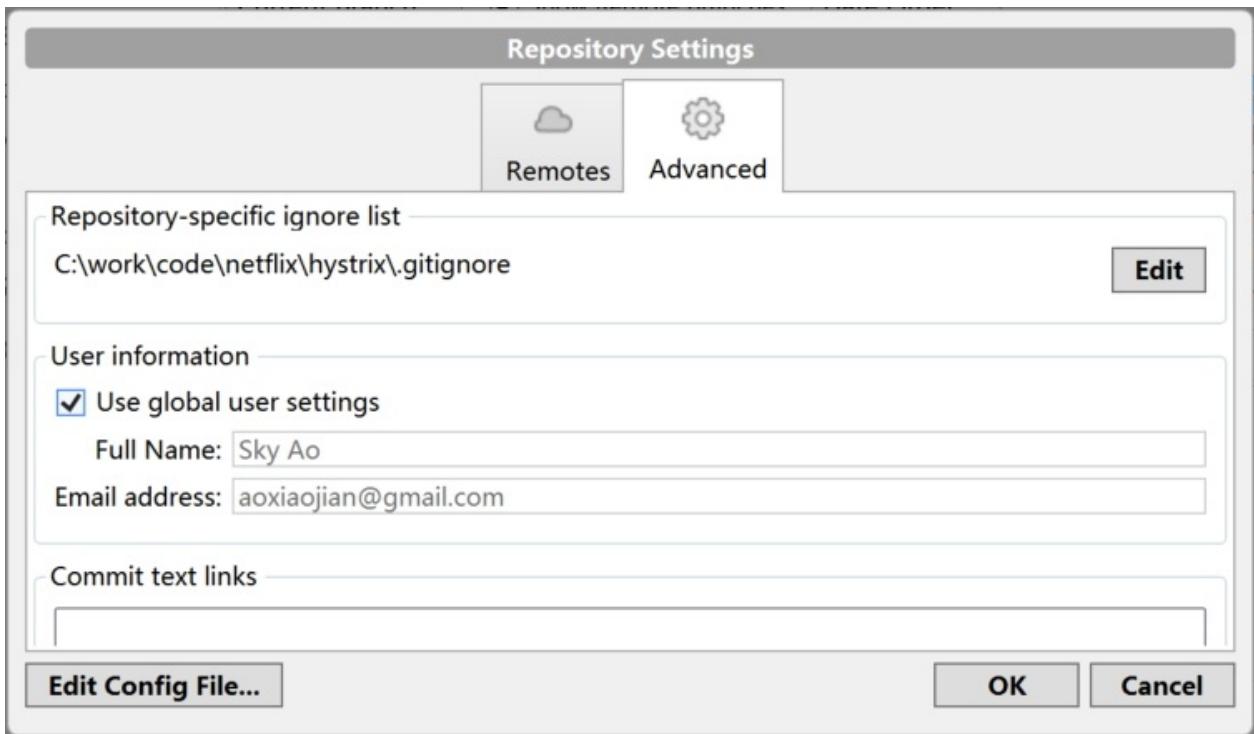
"Options" 中可以设置默认用户信息，如图：



一般推荐在这里设置，如果某些仓库有特殊需要再在仓库级别单独设置。

设置仓库级别的用户信息

在菜单中打开 "Repository" -> "Repository Settings" -> "Advanced" :



去掉"Use Global user settings"前面的勾选，就可以修改当前仓库级别的用户信息了。

设置 SSH

背景知识介绍: git 支持两种通讯方式，https和SSH：

- https: URL类似"https://github.com/skyao/leaning-git.git",这种方式要每次输入用户名密码

(当然有些软件可以帮你记住用户名密码)

- SSH: "git@github.com:skyao/leaning-git.git"，这种方式只要设置好SSH key，每次操作不用再输入用户名密码

使用 OpenSSH 作为客户端

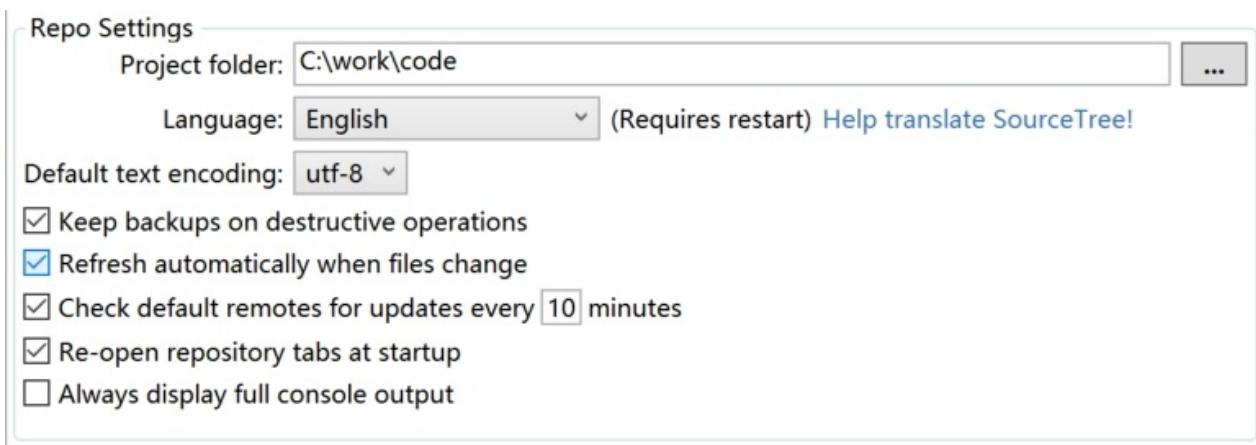
选择 OpenSSH 作为SSH客户端，然后SSH Key指向已经生成好的id_rsa：



使用 PuTTY 作为客户端

需要将 id_rsa 转一次得到 PuTTY key file，比较麻烦，一般推荐用 OpenSSH。

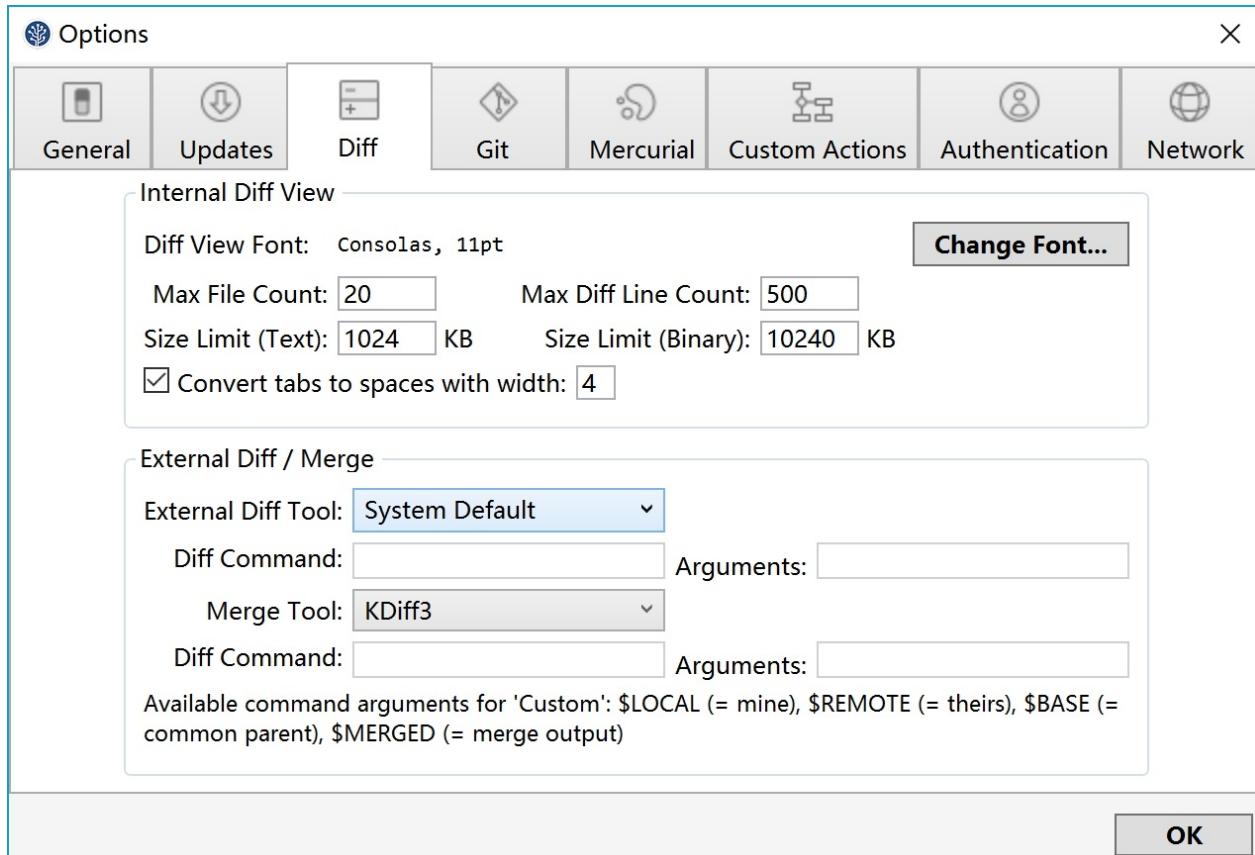
仓库设置



- Project folder: 设置默认的项目文件夹，在新建仓库/克隆仓库等操作时默认就会使用这个路径来保存新的仓库，如果不设置则windows下默认为当前用户的Documents目录。
- Language: 修改 sourcetree 界面的语言,推荐还是用英语吧，各种术语保持一致，用中文还需要中文英文对应过来。
- Default text encoding: 默认utf-8，一定不要改

Diff 设置

Diff设置中，"External Diff/Merge" 用来设置外部的diff和merge工具，默认都是"System Default"。下拉框中可以选择支持的外部的diff和merge工具，比如merge tool我设置为kdiff3了。



注意选择的外部工具是需要自己另行安装的。

kdiff3

先在 kdiff3 的网站下载最新版本的 kdiff3：

<https://sourceforge.net/projects/kdiff3/files/>

然后安装。

仓库

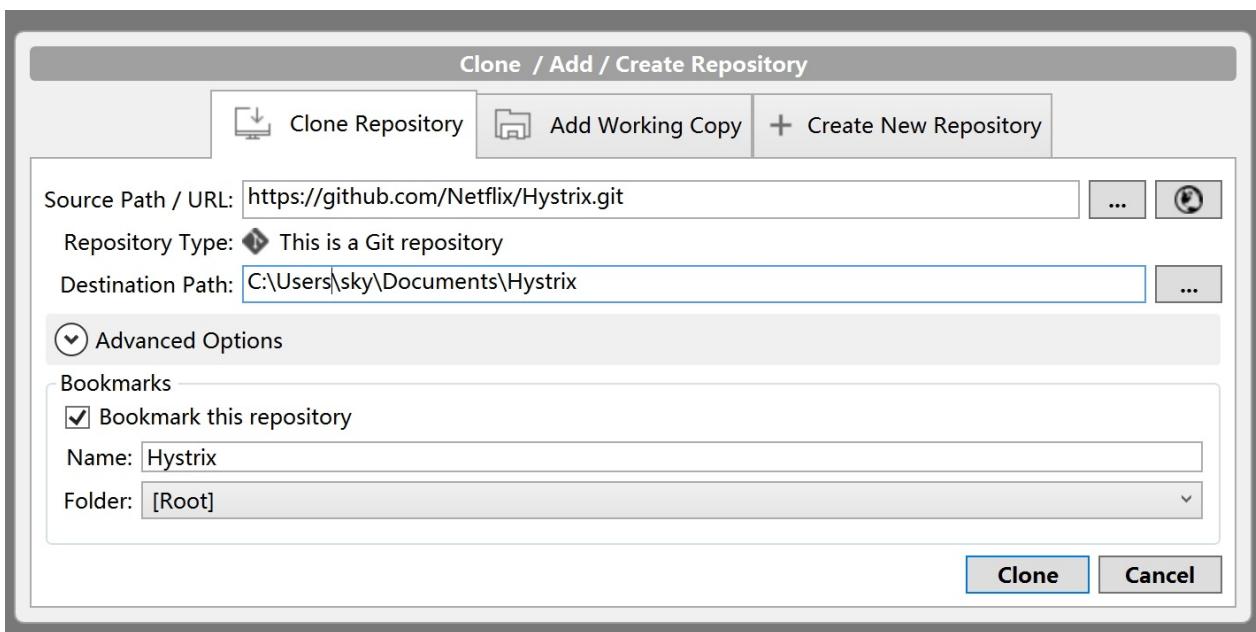
在打开的 SourceTree 中点左上角的"克隆/新建"：



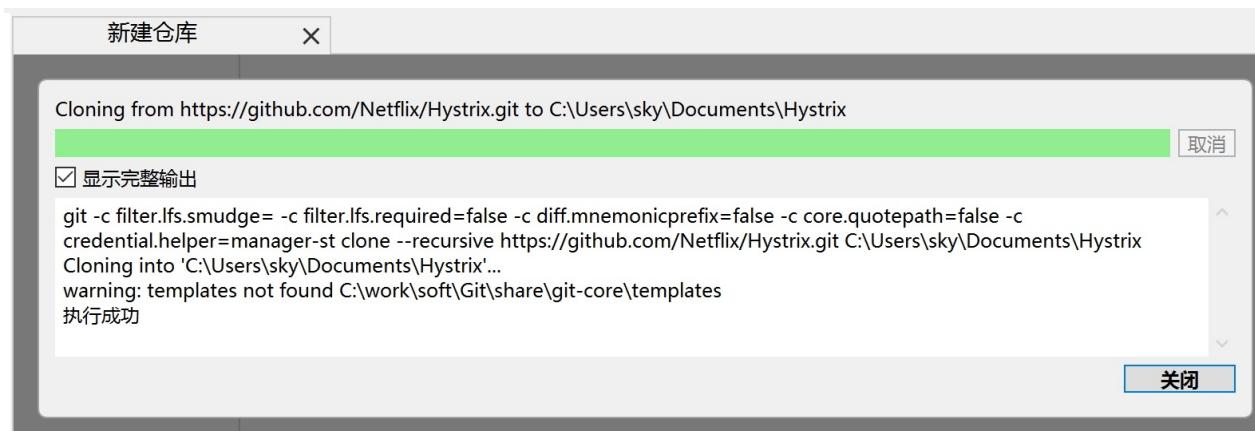
克隆

选择"clone repository",这个选项是从远程仓库克隆一份到本地：

使用 **HTTPS** 克隆

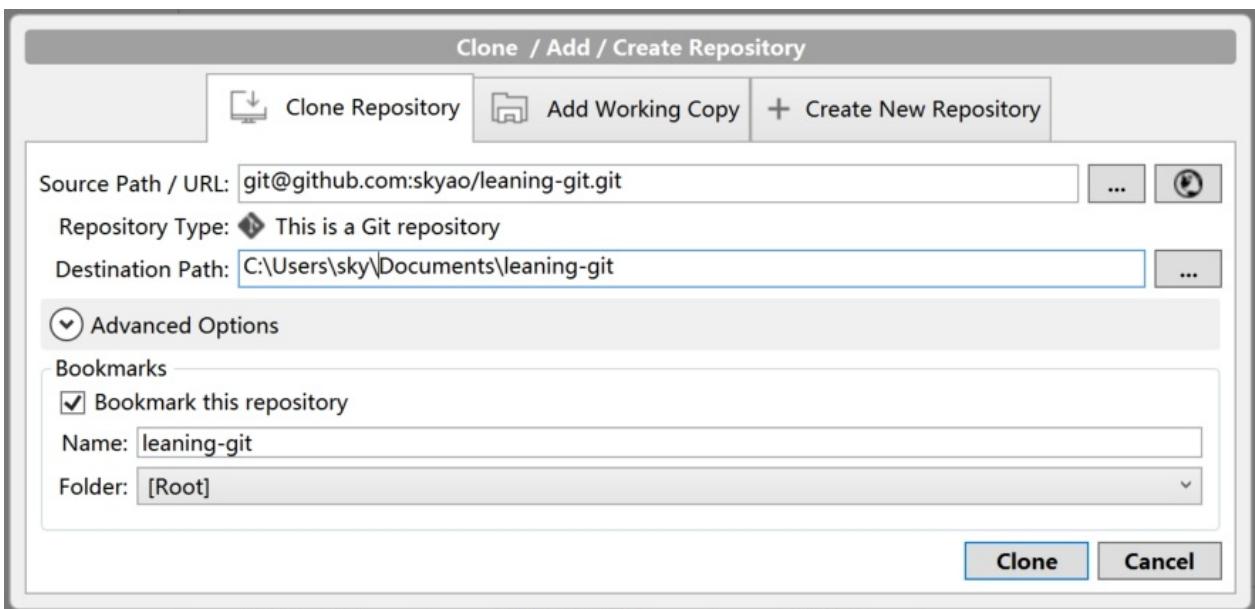


克隆成功之后的提示：



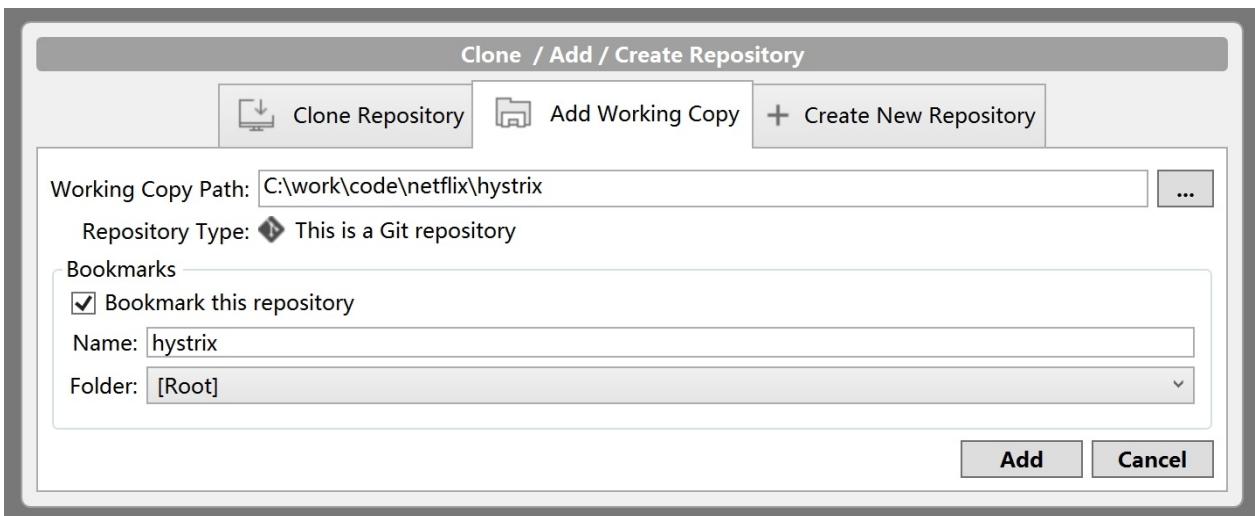
使用 SSH 克隆

如果设置好了SSH客户端，就可以用ssh克隆的方式：

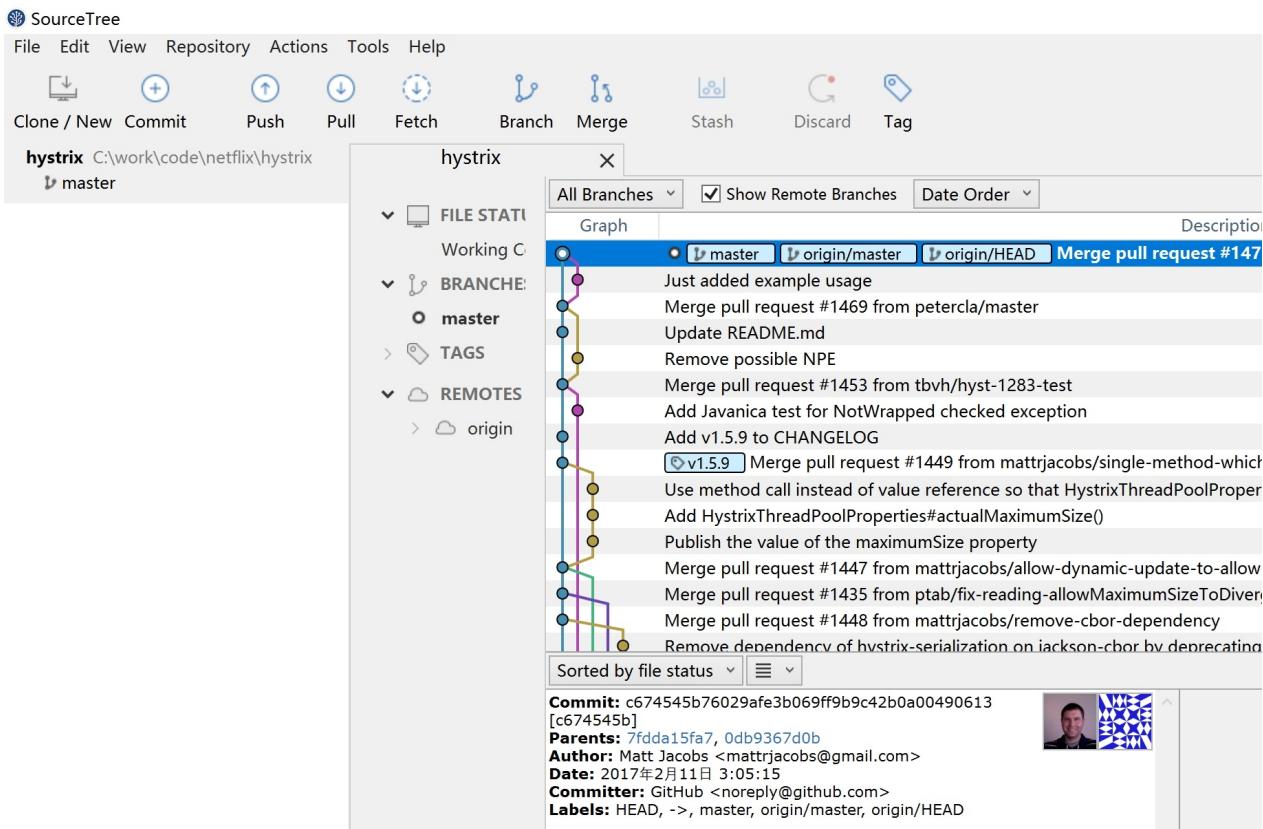


添加工作副本

选择"add working copy",这个选项是本地已经有一份克隆好的仓库，现在只是让 sourcetree 打开这个已经克隆好的仓库：



以下为打开的仓库视图：



分支 / branch

查看分支

这里可以看到当前仓库的分支情况：

Pull Fetch Branch Merge Stas

hystrix

FILE STATUS
Working Copy

BRANCHES
master ←

TAGS

REMOTES
origin
1.4.x
2.0.x
contributing
gh-pages
HEAD
master

All Branches

Graph

Sorted by fi

Commit: c67 [c674545b]
Parents: 7fc

- 红色箭头处是当前所在的本地分支
- 红圈内是远程仓库的所有分支

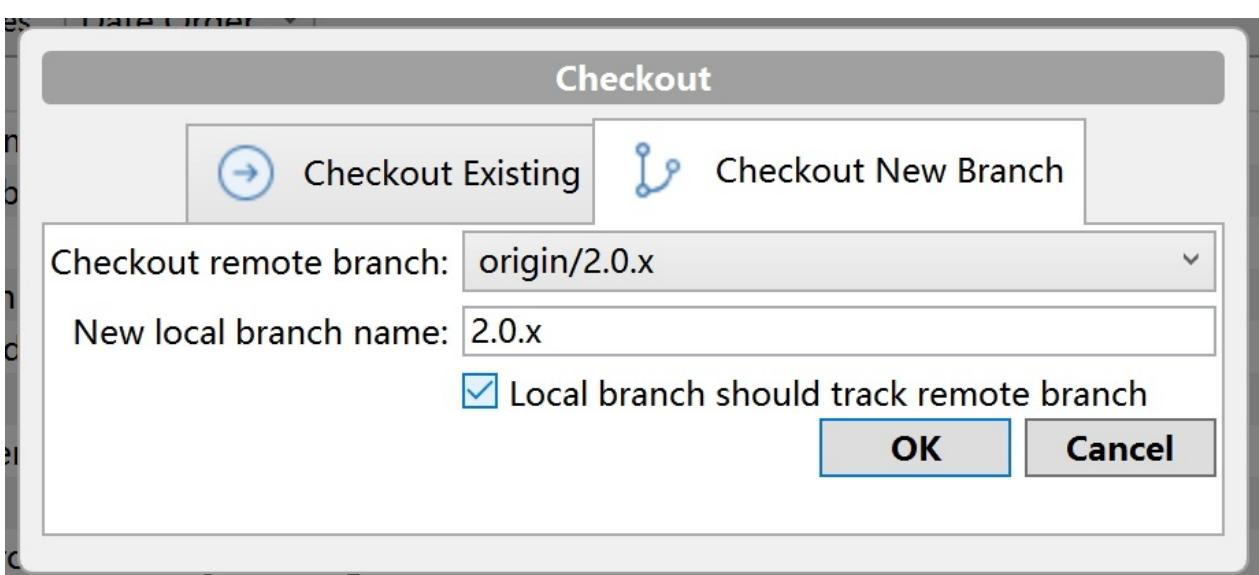
可以对比一下命令行下的情况，`git branch` 命令是显示本地分支，前面带“*”的是当前分支，`git branch -a` 是显示包含本地分支和远程分支的所有分支：

```
sky@skyxps MINGW64 /c/work/code/netflix/hystrix (master)
$ git branch
* master

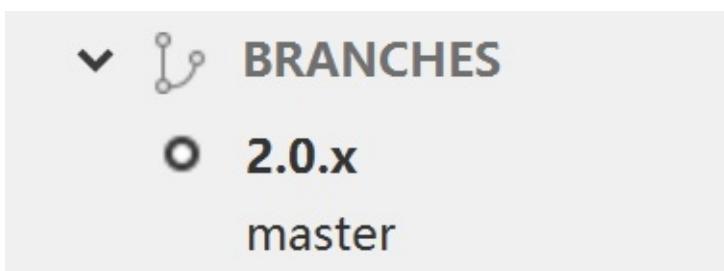
sky@skyxps MINGW64 /c/work/code/netflix/hystrix (master)
$ git branch -a
* master
  remotes/origin/1.4.x
  remotes/origin/2.0.x
  remotes/origin/HEAD -> origin/master
  remotes/origin/contributing
  remotes/origin/gh-pages
  remotes/origin/master
```

checkout 分支

在 REMOTES 远程分支这里，找到要 checkout 的分支，右键，点击“checkout origin/###”



checkout 成功之后，当前本地的分支会切换到你新checkout的分支上：



注意此时本地就两个分支，前面带圈的是当前所在的分支。

同样可以对比一下命令行下的情况：

```
$ git branch
* 2.0.x
  master

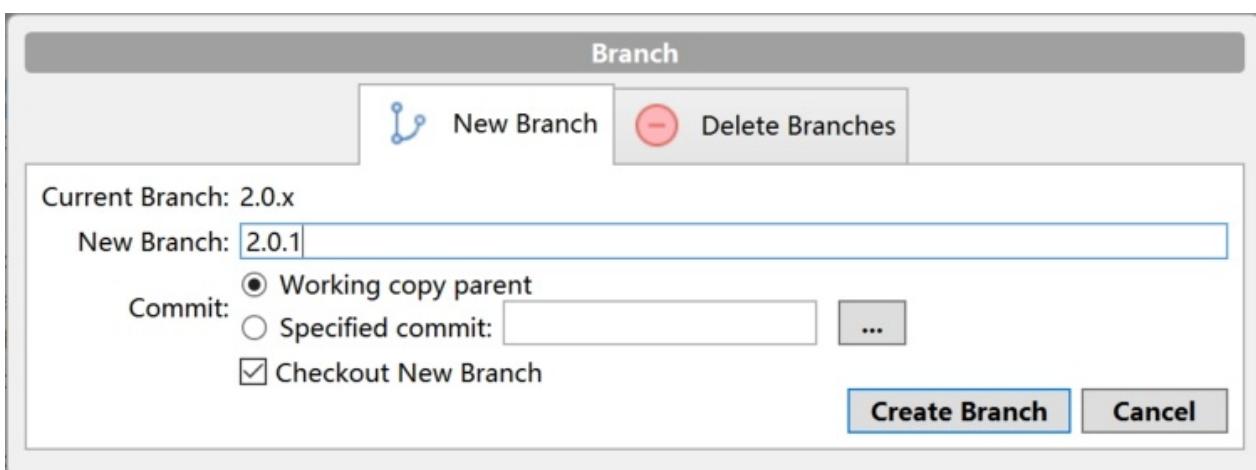
sky@skyxps MINGW64 /c/work/code/netflix/hystrix (2.0.x)
$ git branch -a
* 2.0.x
  master
  remotes/origin/1.4.x
  remotes/origin/2.0.x
  remotes/origin/HEAD -> origin/master
  remotes/origin/contributing
  remotes/origin/gh-pages
  remotes/origin/master
```

创建新分支

点击工具栏中的 "Branch" 按钮，

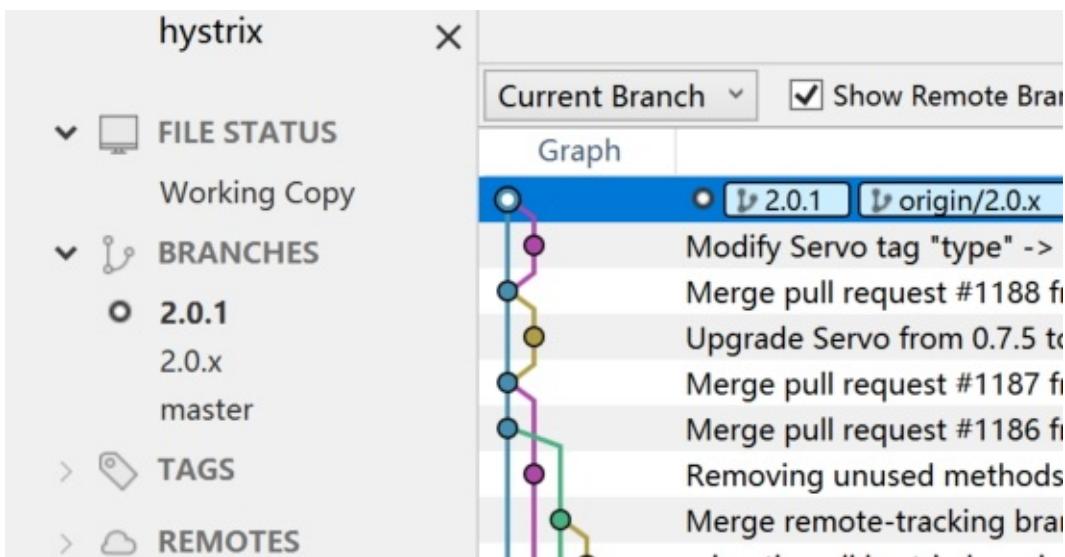


或者菜单中的 "Repository" -> "Branch...", 输入新分支的名字：

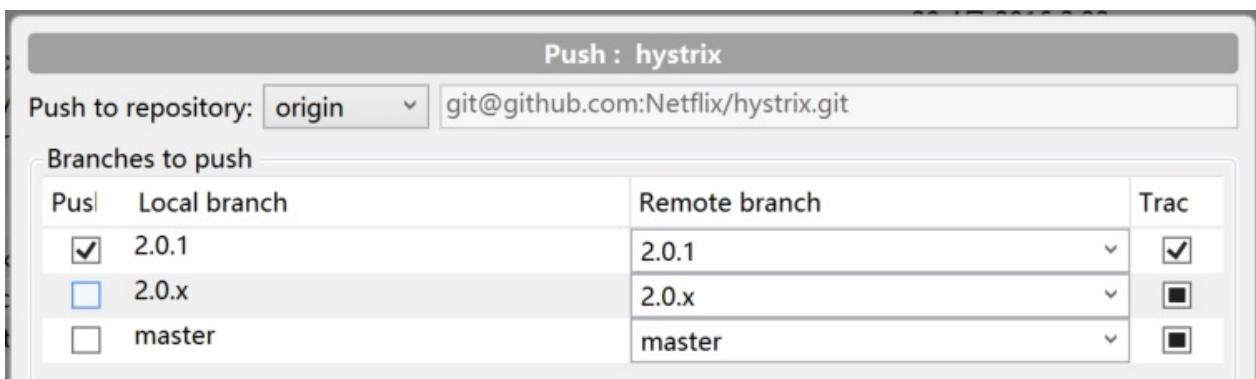


这表示从当前本地分支所在的最后一个commit开始拉出一个新的branch(当然也可以选择"Specified commit"然后选择某个特定的commit)，"Checkout New Branch"勾选表示新分支创建成功之后就直接checkout到这个新建的分支。

点"Create Branch"，成功之后，就已经转到新的branch上去了：



新创建的 branch 目前只存在于本地，需要 push 到远程仓库，在该branch上右键，选择"Push to" -> "origin"：



点"Push"就可以push到远程仓库。

删除分支

在本地分支上右键，选择"Delete ####" 即可删除该分区。

注意在删除该分支前，先checkout到其他分支，否则会删除失败。

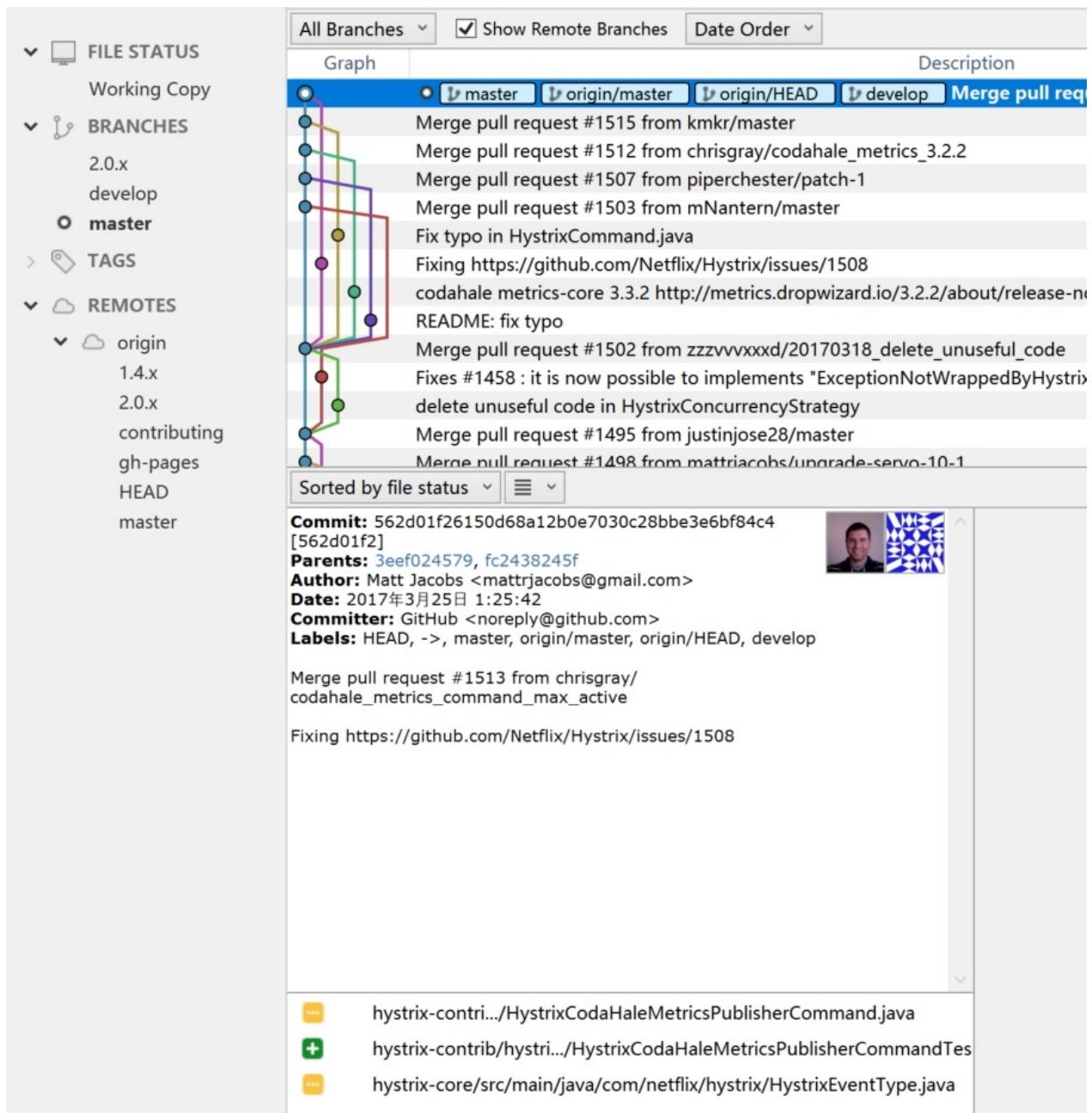
这个方式甚至可以直接操作远程分支，直接删除远程仓库中的分支，不过建议谨慎使用。

查看提交信息

打开仓库，checkout分支之后，就可以看到提交情况。

概况

如图，每一行代表一个提交/commit：



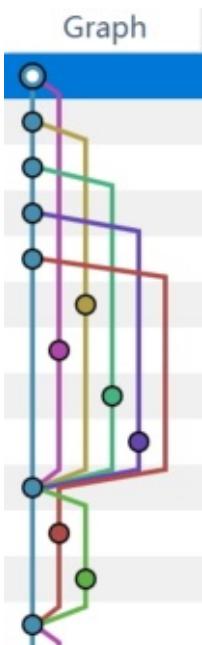
默认是按照时间排序(Date Order)，排在最上面的是最新的提交。可以修改为"Ancestor Order"/按照祖先排序。

默认是"All Branches"，所有分支的提交都会显示在这个界面上。可以在下拉框中选择"Current Branch"只看当前分支(和从当前分支拉出去再合并回来的分支)

"Show Remote Branches"选择是否显示远程分支的提交。(TBD:貌似没有看到有什么差别后作用，后续再更新)

查看 Graph

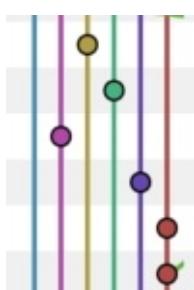
左边竖直的 Graph 是每个提交在各自 branch 上的一个示意图：



- 每行都有一个圆点，代表一次提交，不同分支的原点颜色不同。
- 坚线代表一个分支，颜色和原点一致。两条平行坚线代表两个不同的分支在并行开发。
- 从一个原点拉出一条新的坚线，表示从该分支的这个位置开始建立了一个新的分支
- 新分支上的一个一个原点，表示在这个新分支上的陆续提交
- 新分支在若干次提交之后，于某处和原分支汇合，代表新分支合并回原分支

我们来看几种典型的开发流程对应的图形：

1. 多分支并行开发：体现在图形上是多条并行线，各自有提交

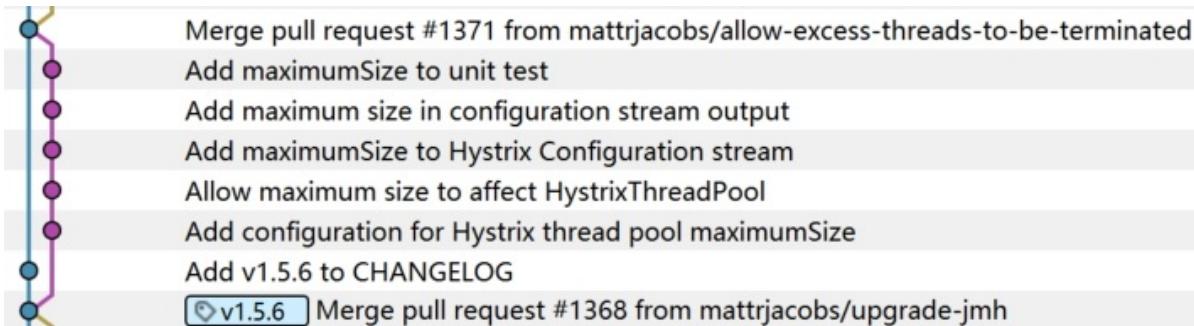


2. bugfix：体现在图形上，从原有branch拉出，提交bugfix之后迅速merge回来

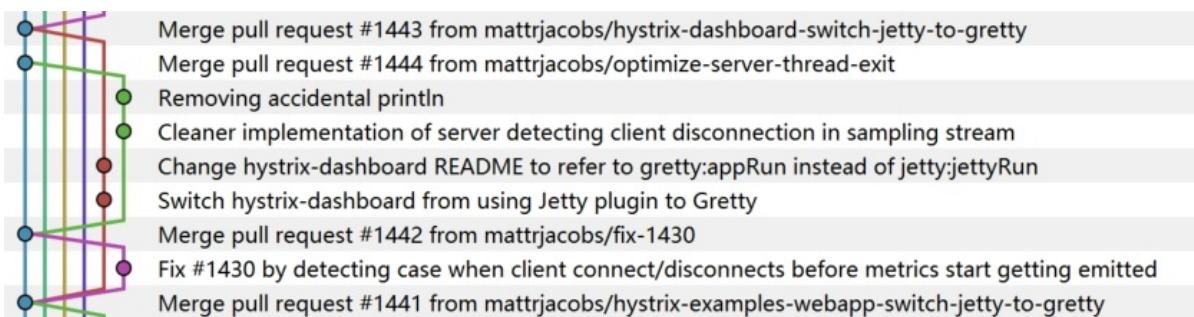


注意右边的提交信息，"Fix"和"Merge...."。

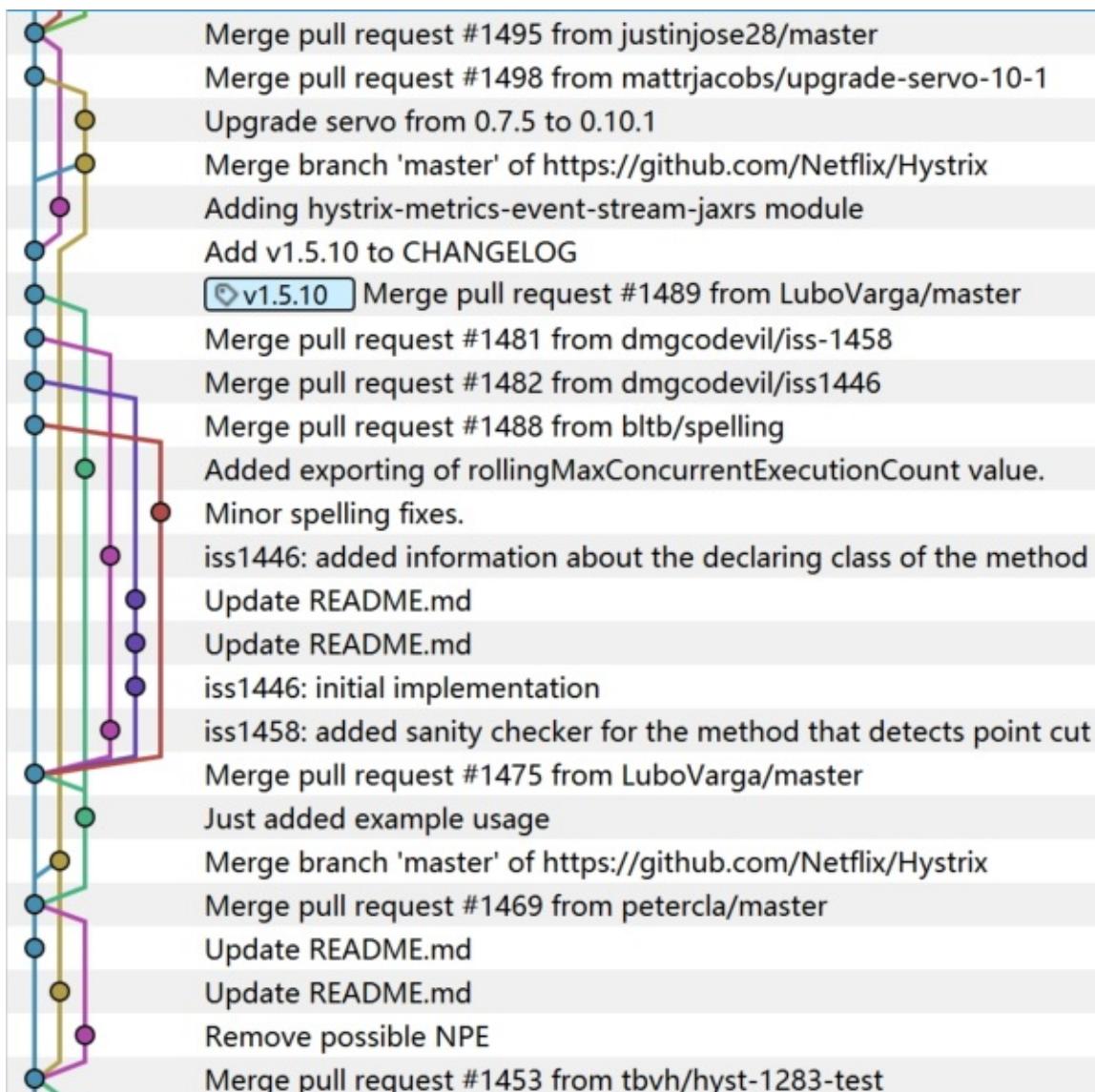
3. feature：体现在图形上，从原有branch拉出，提交多次，然后feature开发之后merge回来原来的branch



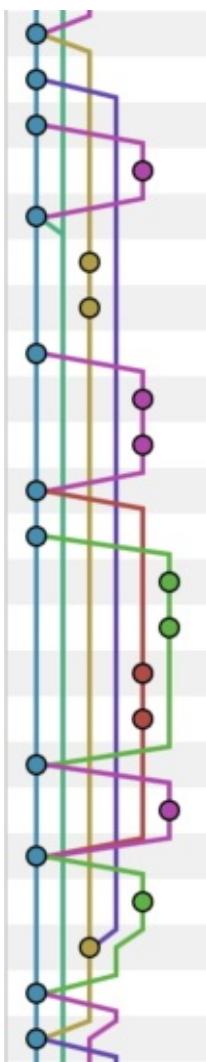
4. 多种方式混合：体现在图形上，先从原有branch拉出做bugfix，然后又拉出做feature，bugfix先merge回来，然后又拉出一个branch做第二个feature。最后两个feature完成再先后合并回来。



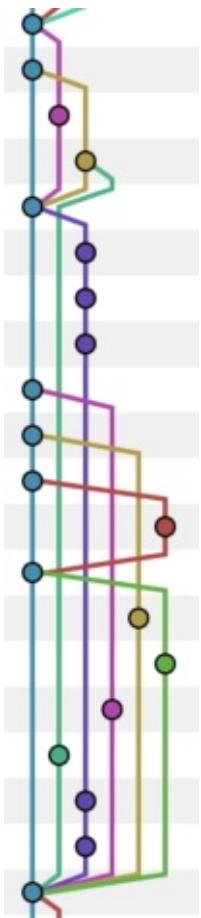
5. 开发中多次merge：看橙色的branch，从master branch拉出后，发现master做了两次提交，然后从master做了一次merge到橙色的branch。在橙色的branch后续开发的过程中，开了好几个branch陆陆续续修改完成再合并回master，这些修改随后做了第二个从master到橙色的branch的merge。最后橙色的branch完成，merge回master。



6. 分支上再拉分支：橙色的branch先拉出来，开发一点后，从橙色的branch拉了一个蓝紫色的分支，然后两个分支分别开发再陆续merge回master。



7. 开分支做merge：青色分支拉出来之后，开发完成，在准备merge回master时，master上已经有比较多的提交。为了避免过多冲突导致master不稳定，从master上临时拉了一个新的橙色分支出来，先完成和青色分支的merge，解决冲突之后，再一起merge回master。



查看提交信息

点击每个提交时，会显示这个提交的信息：

Sorted by file status ▾

Commit: 562d01f26150d68a12b0e7030c28bbe3e6bf84c4
[562d01f2]
Parents: 3eef024579, fc2438245f
Author: Matt Jacobs <mattrjacobs@gmail.com>
Date: 2017年3月25日 1:25:42
Committer: GitHub <noreply@github.com>
Labels: origin/master, origin/HEAD, master, develop

Merge pull request #1513 from chrisgray/
codahale_metrics_command_max_active

Fixing <https://github.com/Netflix/Hystrix/issues/1508>

- 📁 hystrix-contri.../HystrixCodaHaleMetricsPublisherCommand.java
- ➕ hystrix-contrib/hystri.../HystrixCodaHaleMetricsPublisherCommandTes
- 📁 hystrix-core/src/main/java/com/netflix/hystrix/HystrixEventType.java

- Commit：每个提交的SHA，如"562d01f26150d68a12b0e7030c28bbe3e6bf84c4"，通常简写为前8位。可以右键点这个提交，然后选"Copy SHA to Clipboard"，复制这个SHA值到

粘贴板。

- Parents：每次提交的parent，对于普通提交，只有一个parent。但是对于merge操作的commit则通常有两个parent。
- Author：作者信息，name + email，我们前面设置的user information显示在这里
- Date：提交的时间。注意是每个commit操作的时间，不是push的时间。
- Committer：TBD 待查
- comments：下面的内容是每次commit时填写的comments

查看文件修改列表

出现在提交信息下方的是本地提交的文件修改列表：

	hystrix-dashboard/build.gradle
	hystrix-dashboard/src/main/ja.../EurekaService.java
	hystrix-dashboard/src/main/.../EurekaServiceInfo.java
	hystrix-dashboard/src/main/java.../Application.java
	hystrix-dashboard/src/main/jav.../Applications.java
	hystrix-dashboard/src/mai.../EurekaInfoServlet.java
	hystrix-dashboard/src/main/java/.../UrlUtils.java
	hystrix-dashboard/src/main/t.../UrlUtilsTest.java
	hystrix-dashboard/src.../EurekaServiceInfoTest.java
	hystrix-dashboard/src/main/webapp/WEB-INF/web.xml
	hystrix-dashboard/src/main/webapp/index.html

文件前面的图标有明确的意义：

- ：文件内容发生更改，但是文件本身的存在性没有变化（即不是新增也没有删除）
- ：增加的新文件
- ：删除的文件

查看文件内容修改

点击文件修改列表中的每一个文件，就会在右边出现该文件的内容修改情况。

文件内容修改通常以下几种情况：

1. 增加内容：表现为绿色背景，行最前面是加号

```

private boolean extendedParentFallback;
+ private final boolean defaultFallback;
  private final JoinPoint joinPoint;

```

2. 删除内容：表现为淡红色背景，行最前面是减号

```

```xml
<dependency org="com.netflix.hystrix" name="hystrix-codahale-metrics-publisher" rev="1.1.2" />
-
\ No newline at end of file

```

3. 修改内容：由于git对内容的处理是以行为单位，因此无法直接体现出来是修改了该行的哪部分内容，在git中对行内容的修改表现为一次删除+一次增加。

典型如下，实际只是修改"3.1.2"为"3.2.2":

```

dependencies {
 compileApi project(':hystrix-core')
- compileApi 'io.dropwizard.metrics:metrics-core:3.1.2'
+ compileApi 'io.dropwizard.metrics:metrics-core:3.2.2'
 testCompile 'junit:junit-dep:4.10'
 testCompile 'org.mockito:mockito-all:1.9.5'

```

下面是另外一个典型，实际该行没有实质修改，只是语法要求最后的";"在增加新的枚举后要修改为",":

```

CANCELLED(true),
-
COLLAPSED(false);
+ COLLAPSED(false),
+ COMMAND_MAX_ACTIVE(false);

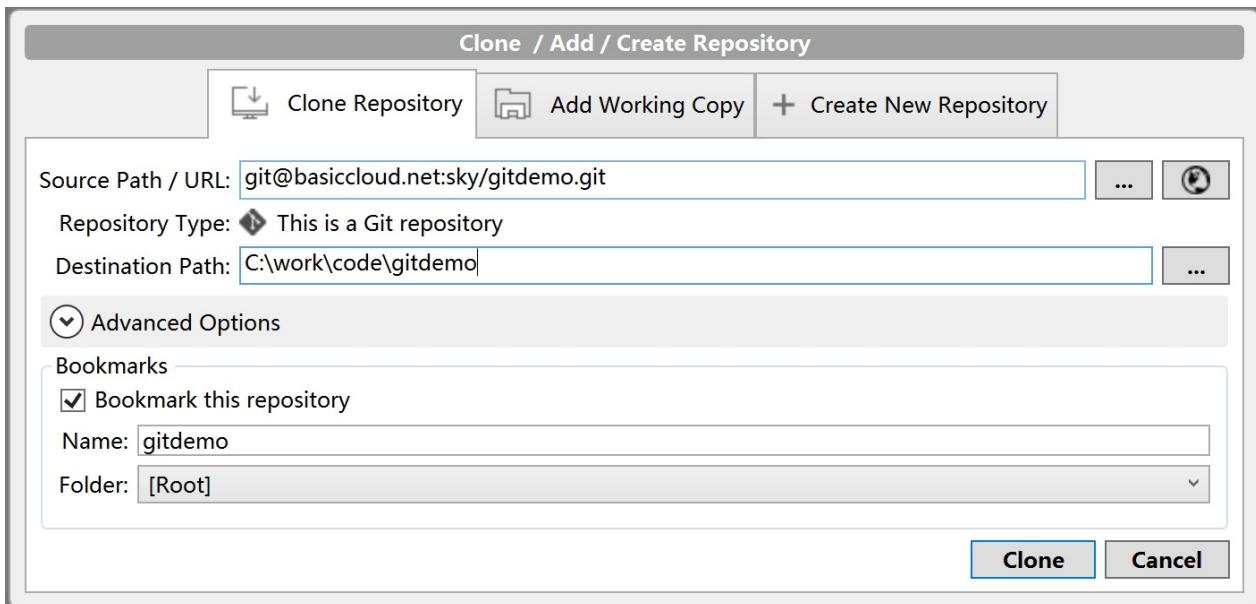
```

# 基本操作

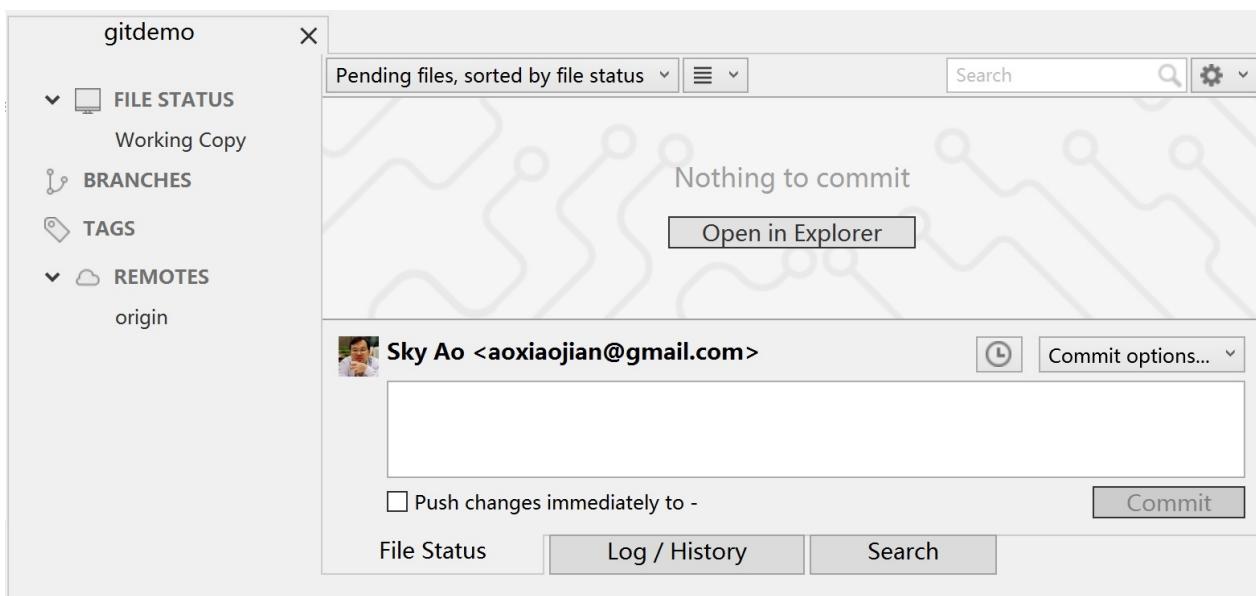
介绍在 sourcetree 下进行git的几个基本操作的方式，顺便熟悉一下 sourcetree 的界面。

背景：我们建立了一个名为 gitdemo 的空仓库，从零开始，展示日常开发中的基本操作。

## 克隆仓库



此时在 sourcetree 中看到的是一个空的仓库，没有任何文件。

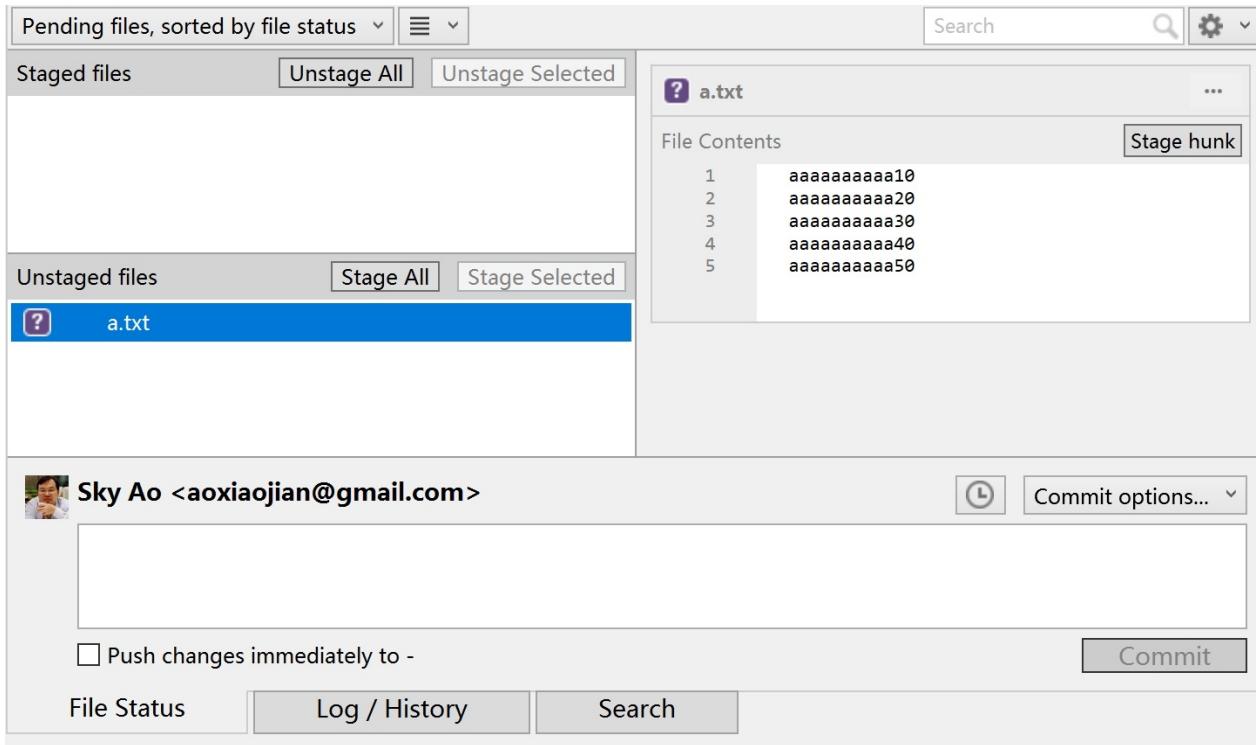


## 新增文件

在本地根目录下新建文件 a.txt, 内容如下：

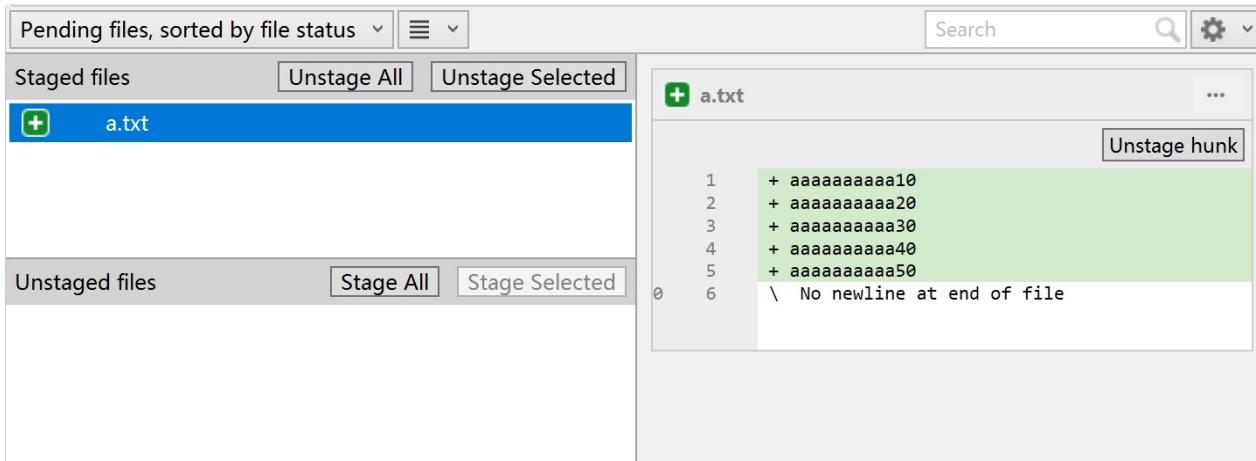
```
aaaaaaaaaa10
aaaaaaaaaa20
aaaaaaaaaa30
aaaaaaaaaa40
aaaaaaaaaa50
```

在 sourcetree 中看到的是：



- "Staged files": 已经修改并stage的文件
- "Unstaged files": 列出的是当前仓库中未被 git 跟踪或者有修改而没有stage的文件，点击每个文件可以在右边看到内容

点击右边的 "Stage Hunk" 按钮可以将当前选中的文件加入到 "Staged files" (或者用"Stage All" / "Stage Selected" 按钮):



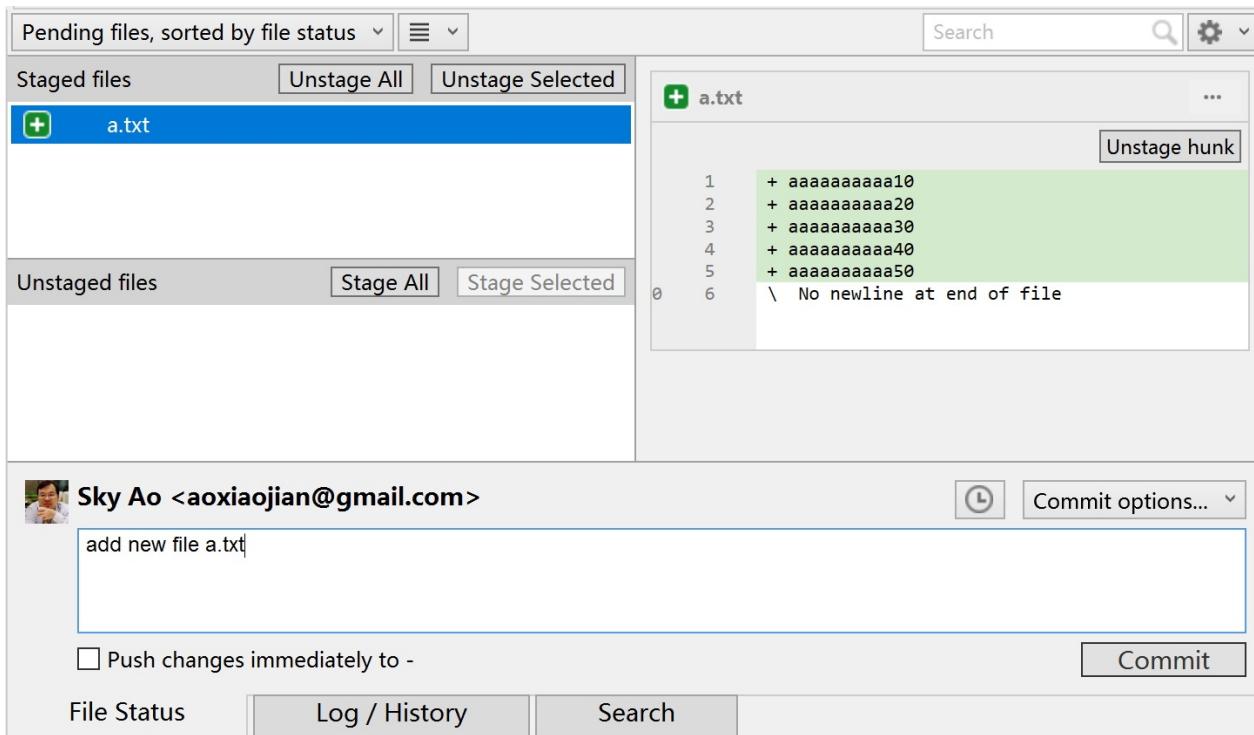
注：等同于命令行下的 "git add a.txt" 命令

如果发现有错误添加文件，可以在 "Staged files" 中选中该文件，然后选"Unstage Selected"，或者右边的 "Unstage hunk"按钮。

同样在点击 "Staged files" 列表中的每个文件时，右边会出现当前这个文件的修改情况。在这里可以非常方便的看到当前已有的文件修改有哪些内容。

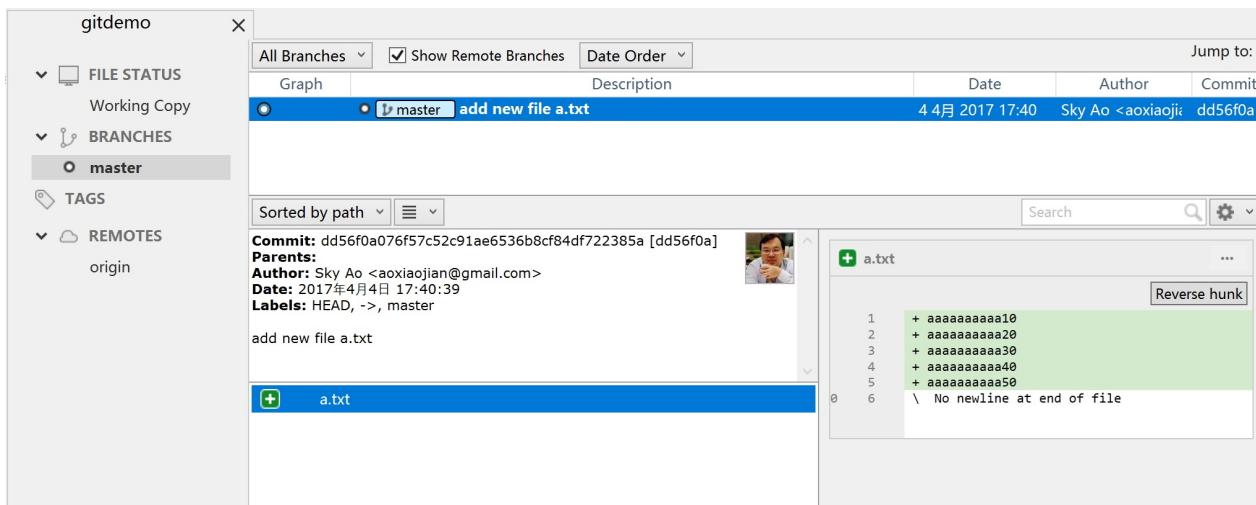
## 提交修改

文件修改完成，就可以提交了，在下方输入当前提交的comments，点击 "Commit" 就可以将 "Staged files" 列表中的所有修改提交：



注：等同于命令行下的 `git commit -m "your comments"` 命令

提交完成后，可以看到本次提交的详细情况：

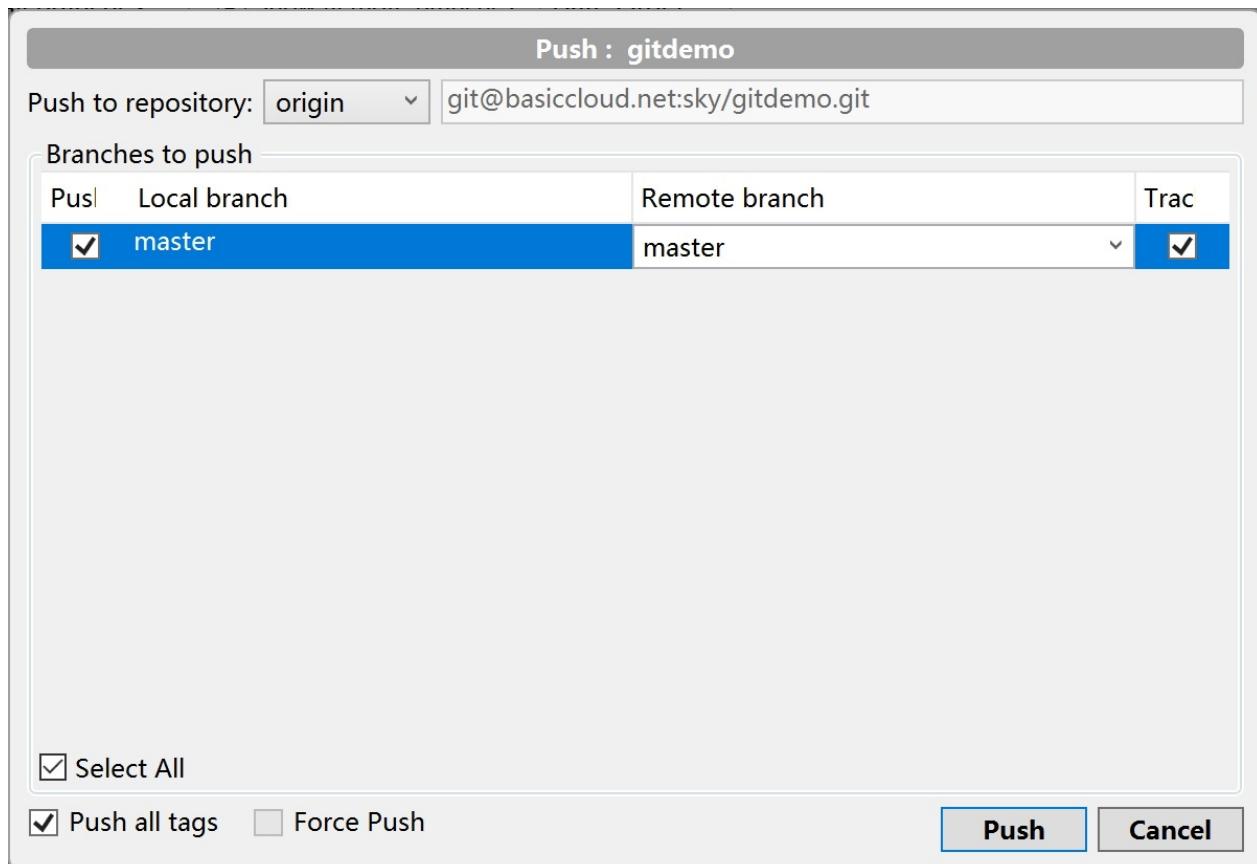


最佳实践：

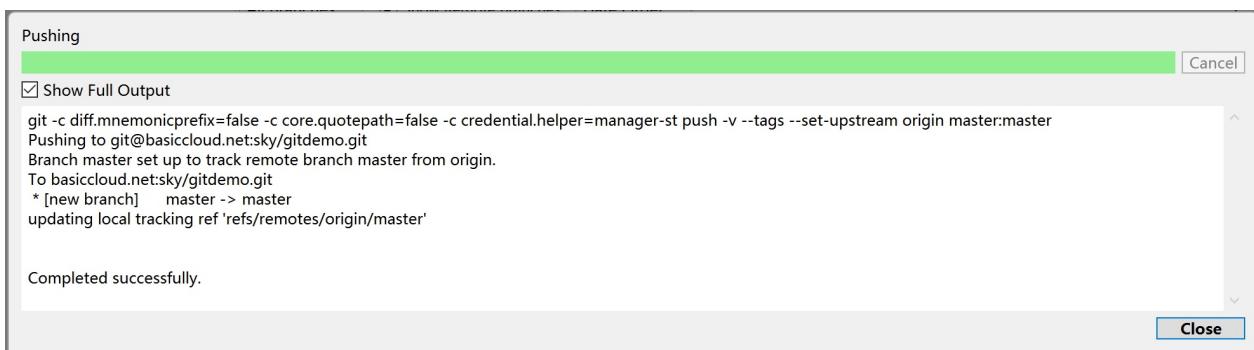
- 每次提交之前，尽量逐个检查 "Stage files" 列表中的每个文件，确保每个文件都是你真的准备提交的，避免因为错误操作提交了不应该提交的文件
- 每次提交之前，尽量逐个检查 "Stage files" 列表中的每个文件，检查每个文件的修改内容是否OK，避免低级失误，比如不小心改动了文件却忘了改回来
- 每次提交的comment内容，务必能准确的描述了这次提交干了什么，以便未来方便浏览

## push/推送

提交完成之后，开始推送到远程仓库，点击工具栏中的 "Push" 按钮(或者"Repository" -> "Push")：

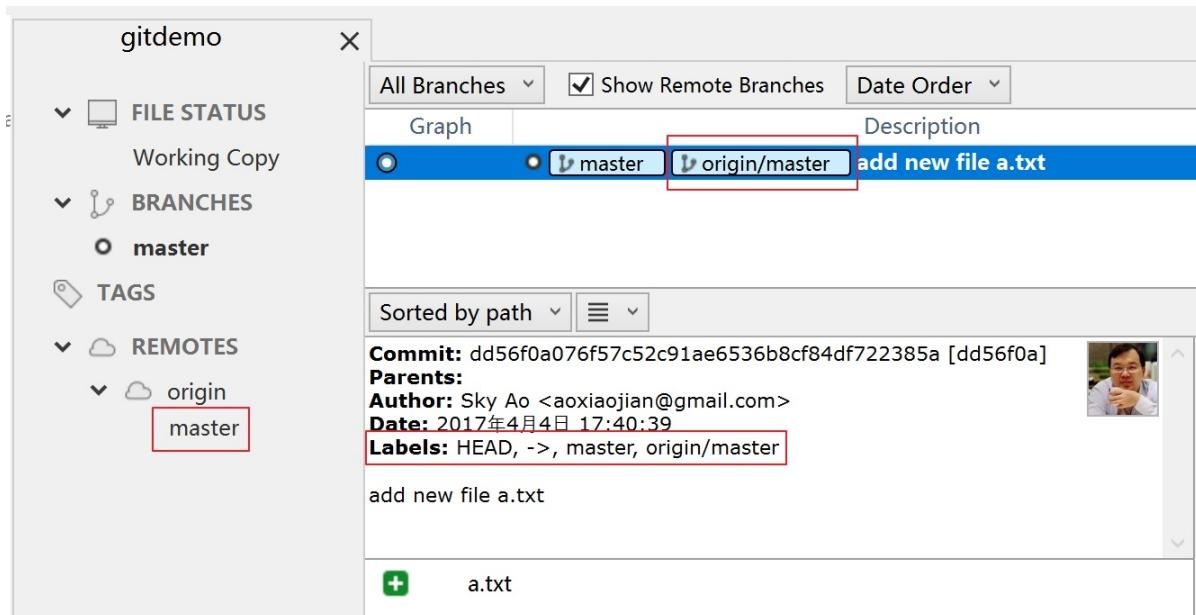


提交完成的情况：

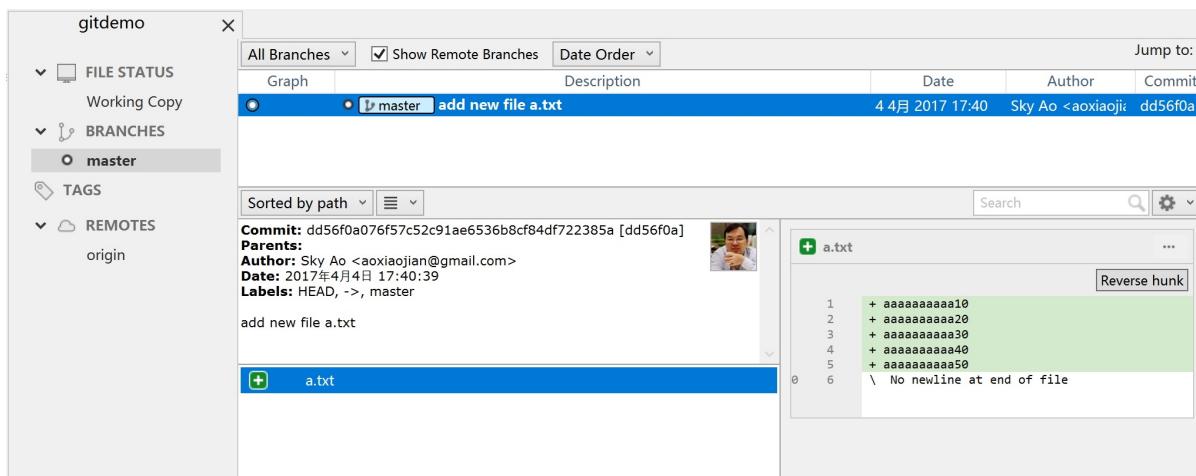


仔细对比 Push 前后这个提交的详细情况：

- push 后：注意三个红色框的内容



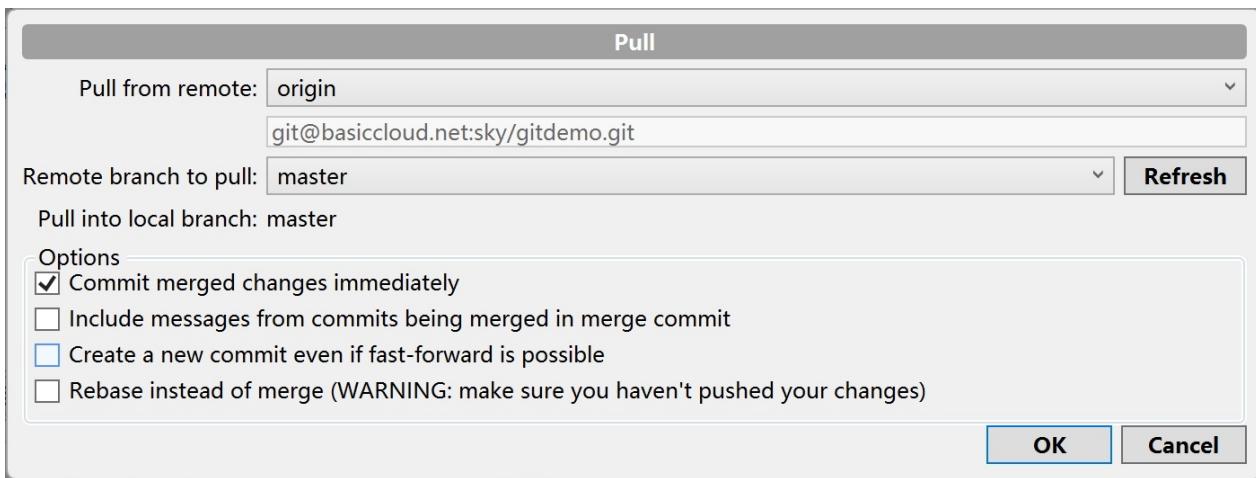
- push 前



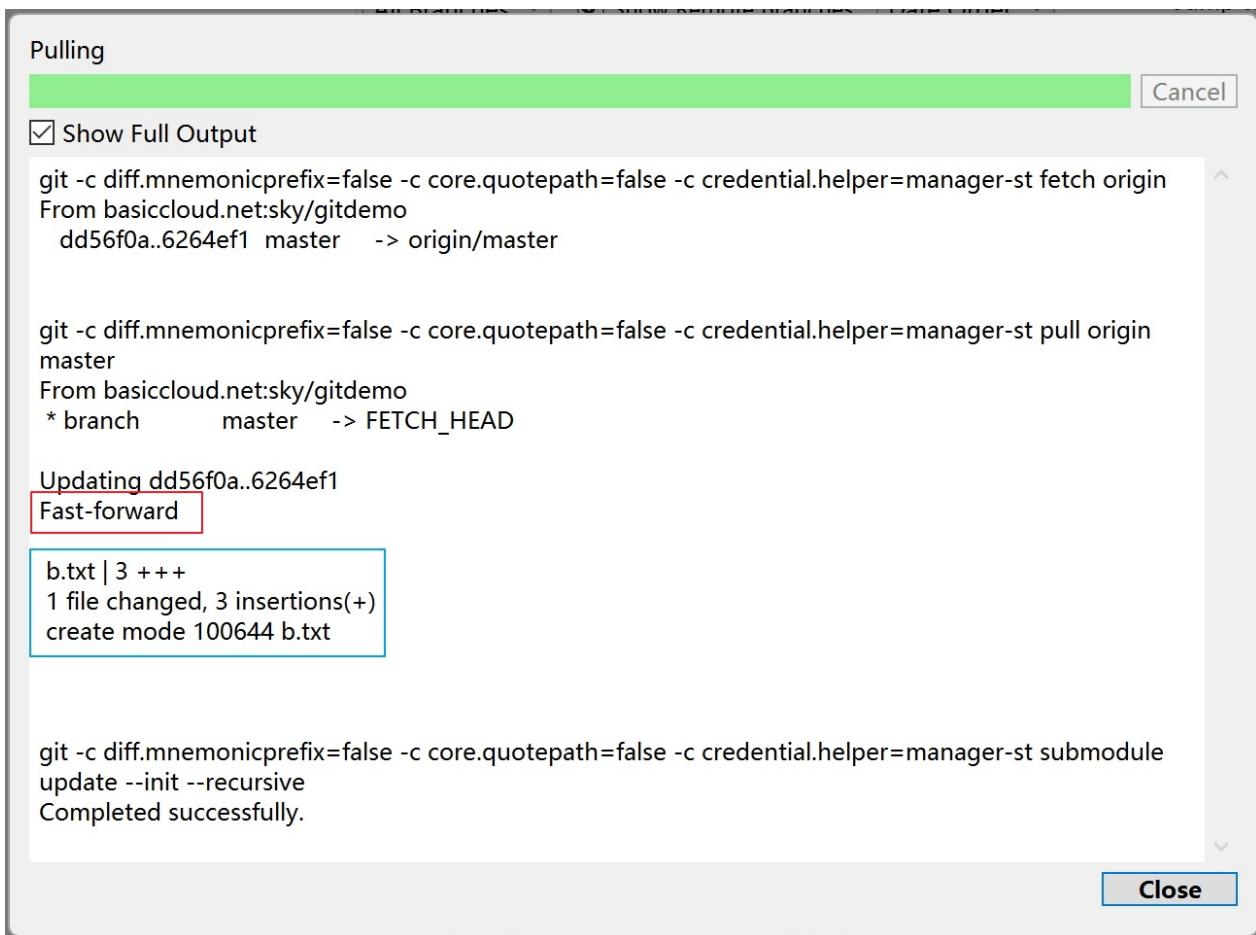
## pull

注：为了演示pull，我们在gitlab的web界面上做了一个增加新文件的操作，增加了文件 b.txt

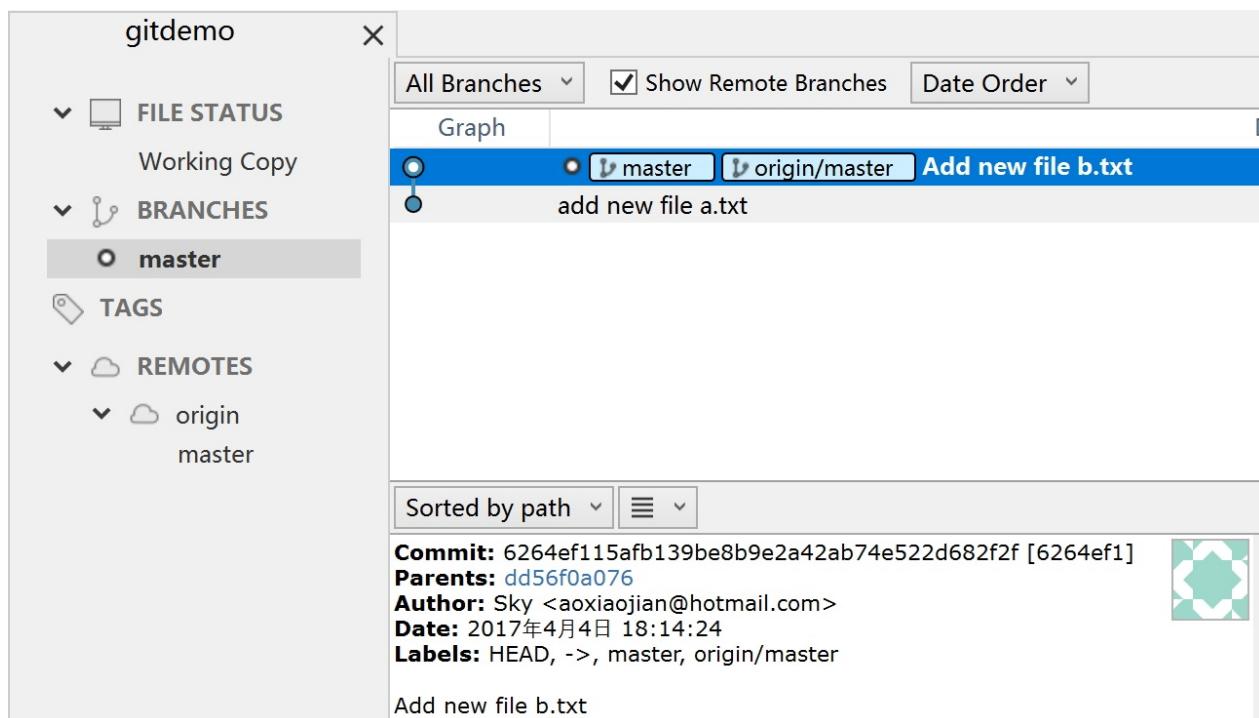
点击工具栏中的 "Pull" 按钮(或者"Repository" -> "Pull")：



为了看到 Pull 的详细情况，我们勾选了"Show Full Output":



在没有冲突的情况下，pull操作很简单的完成了，远程仓库的新的提交被pull下来，体现在sourcetree中就是提交列表中增加了一个新pull下来的提交。然后本地分支的内容得以更新，体现为本地文件中增加了一个b.txt文件：



# Pull 时冲突

## 背景

对于 pull 操作，一种常见的可能就是 pull 下来时发现远程仓库的代码和自己的本地的代码冲突了。

这种事情通常发现在自己修改代码提交并 push 到远程时，时间线一般是这样：

- A pull 最新代码
- A 开始修改
- B pull 最新代码
- B 开始修改
- B 提交并 push 到远程仓库，成功
- A 完成修改，本地提交，然后试图 push 到远程时失败，因为远程已经有修改，不能 fast forward
- A 不得不先进行 pull 操作
- A/B 修改了同一个内容，造成冲突

## 模拟操作

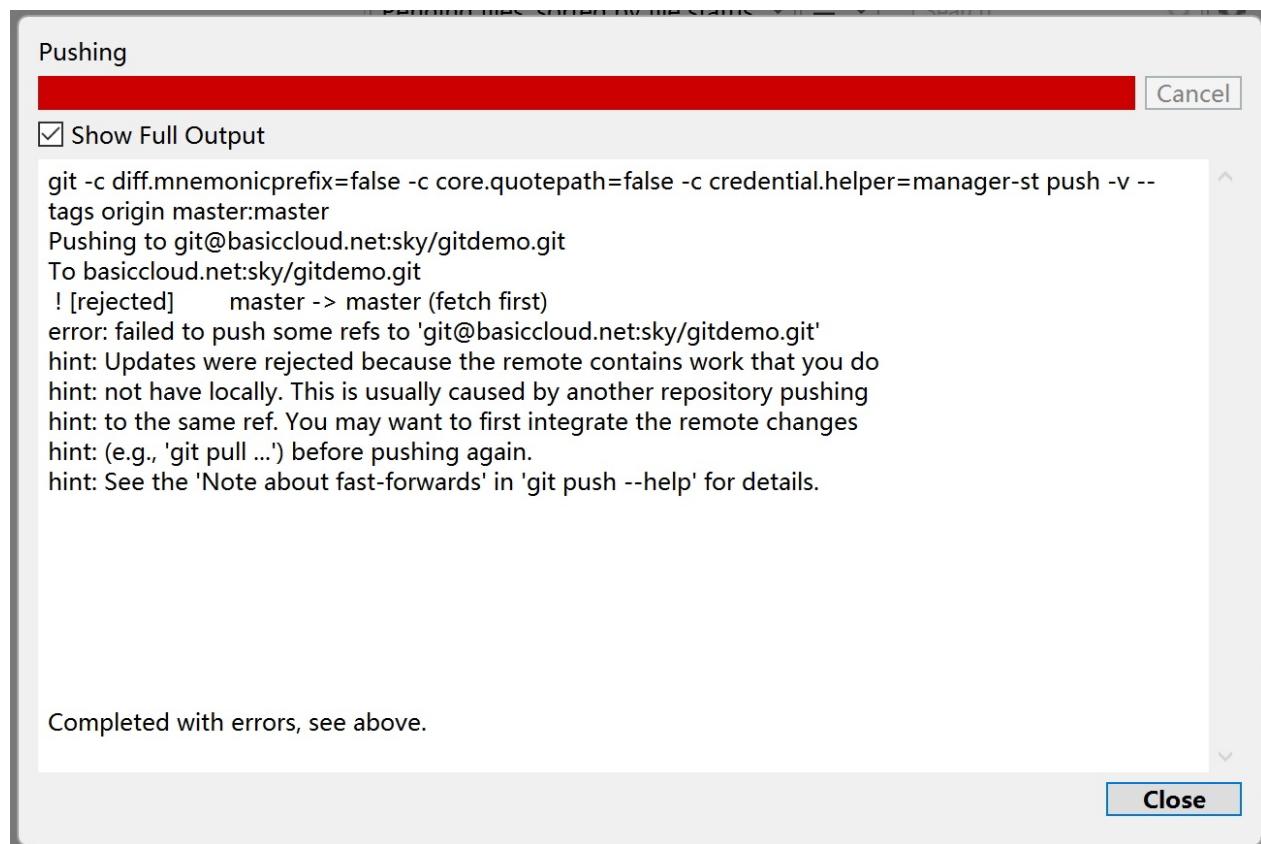
我们模拟同时修改 `a.txt` 文件的场景，假设有其他人修改了该文件的第一行然后提交并 push 到远程仓库：

```
aaaaaaaaaa10 - changed by others
```

但是我们不知情，继续在本地修改 `a.txt` 文件，也是修改第一行：

```
aaaaaaaaaa10 - changed by myself
```

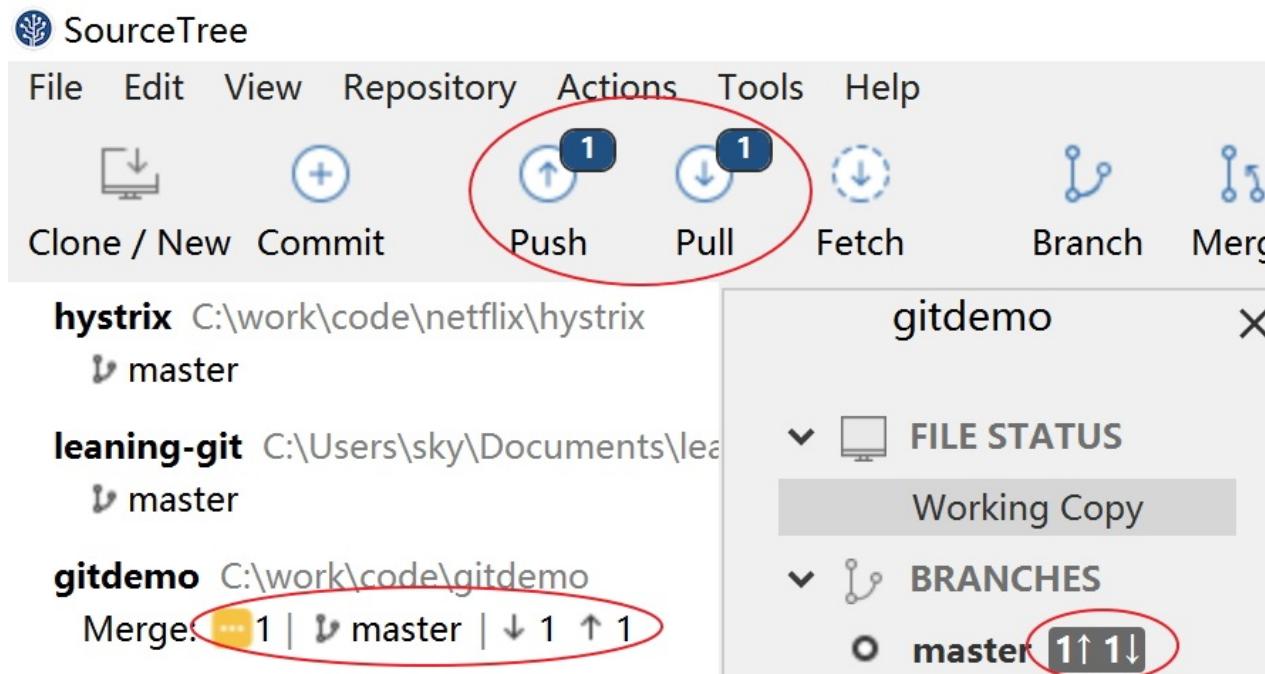
随即 commit 并 push，报错如下：



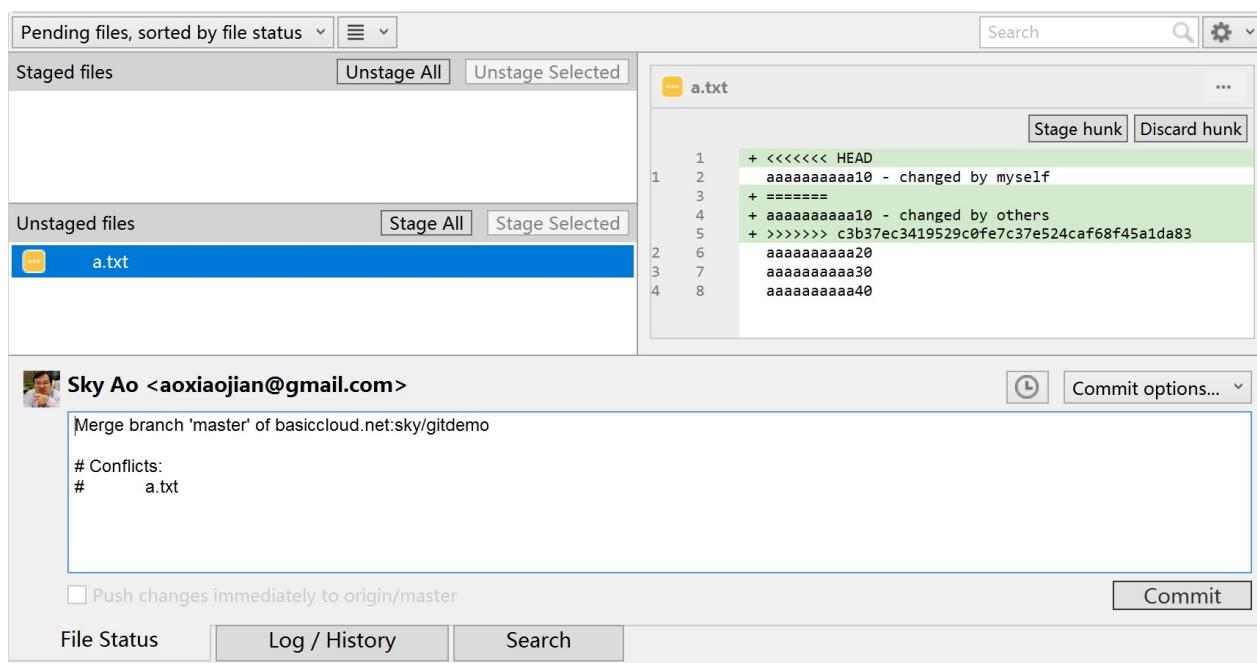
push 操作被拒绝，"Updates were rejected because the remote contains work that you do not have locally"，提示 "You may want to first integrate the remote changes (e.g., 'git pull ...') before pushing again."

## pull 时 merge 冲突

此时只能 pull 了，pull 时 git 会进行一个自动 merge，这里因为我们故意修改了同一行内容，因此必然发生冲突，提示：



点下方的 "File Status" 可以看到详细的文件合并情况：



在这里可以看到，**a.txt** 的自动合并失败，此时 **a.txt** 文件的内容会变成这个样子：

```
<<<<< HEAD
aaaaaaaaaa10 - changed by myself
=====
aaaaaaaaaa10 - changed by others
>>>>> c3b37ec3419529c0fe7c37e524caf68f45a1da83
aaaaaaaaaa20
aaaaaaaaaa30
aaaaaaaaaa40
aaaaaaaaaa50
```

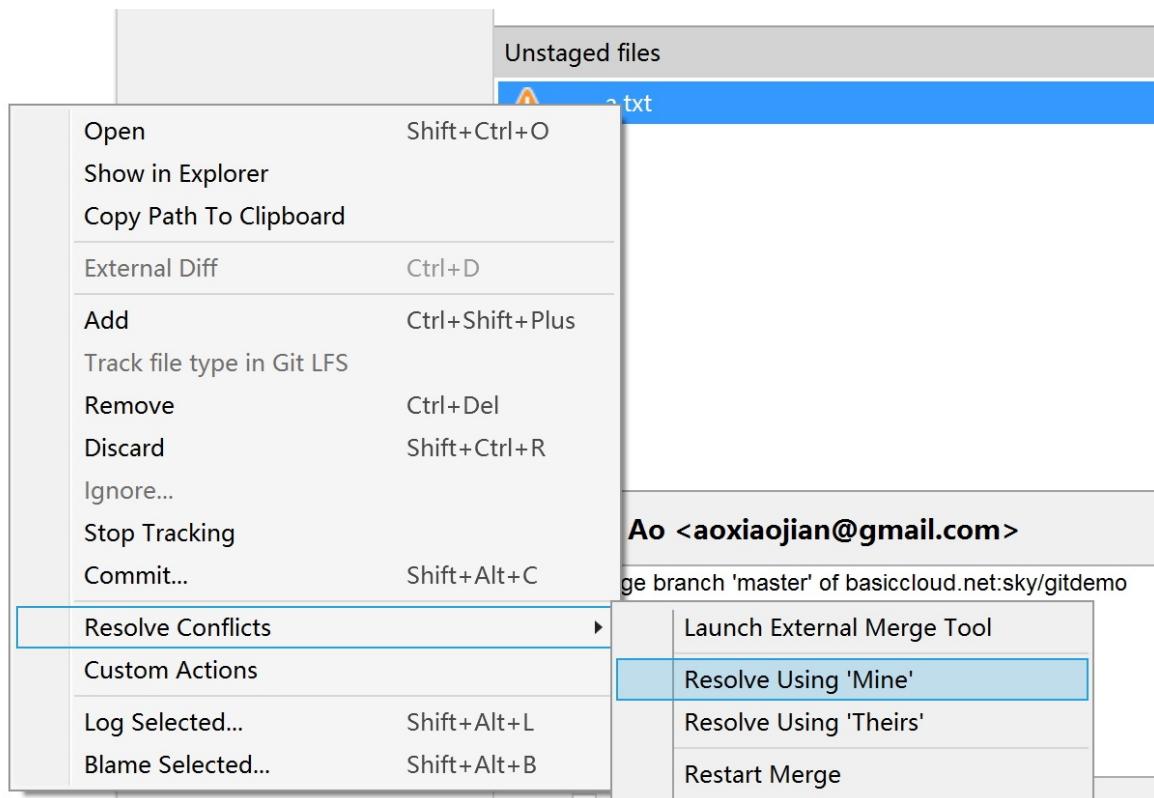
这里的规则：

1. "======" 是分隔符
2. "<<<<< HEAD" 和 分隔符 之间的内容是自己本地打算提交的内容
3. 分隔符 和 ">>>>> SHA" 之间的内容是远程已经提交的内容，可以通过这次提交的 SHA 找到具体的提交信息

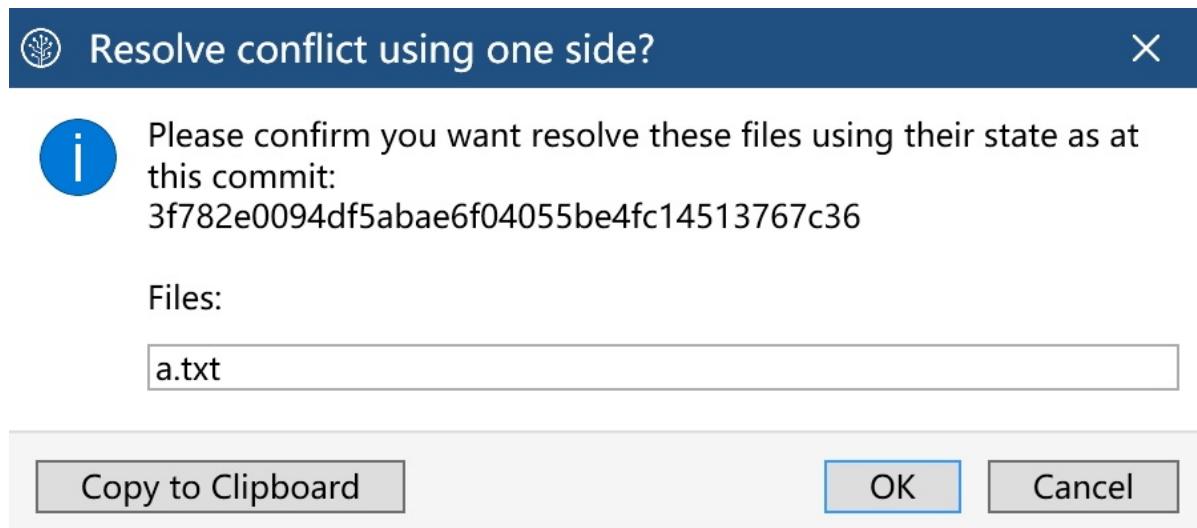
我们现在需要做的事情，是决定此处的冲突应该如何解决，即最后这行的内容应该是什么？

通常来说，选择有三种：

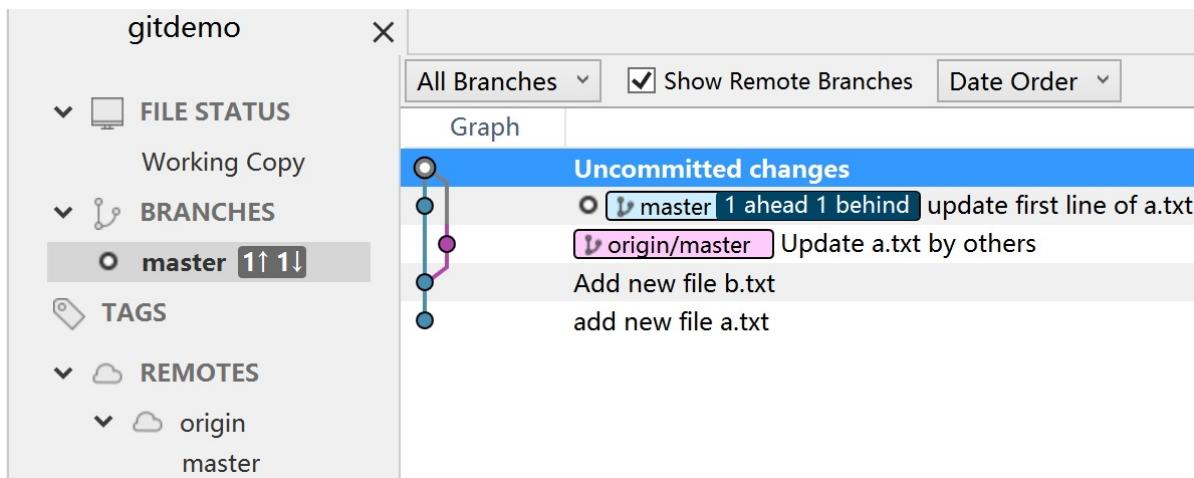
1. 覆盖远程的修改，维持自己的内容



如图所示，在文件上右键，选择"Resolve Conflicts" -> "Resolve Using Mine"，弹出提示：

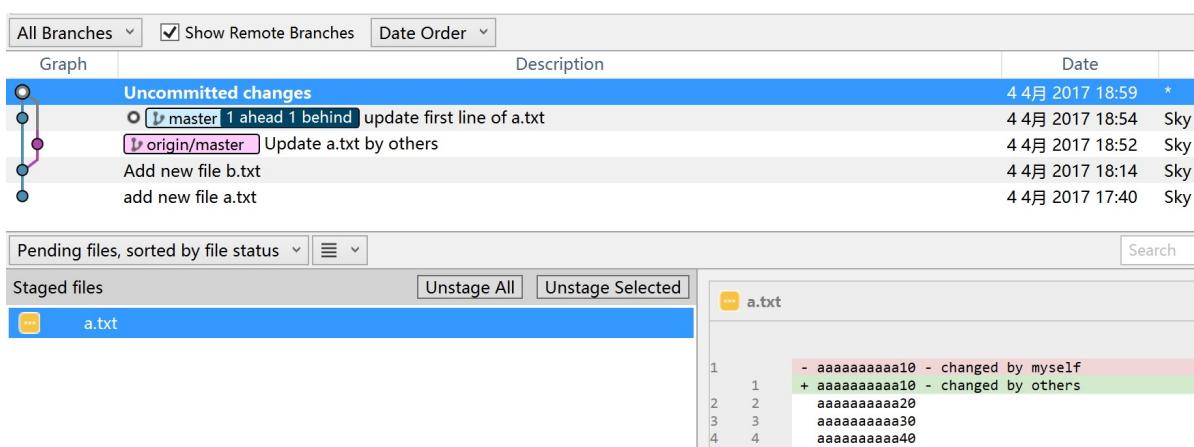


确认即可，此时看到的本地文件状态如图，由于放弃了远程修改维持本地提交的内容，因此表现为本地文件没有修改，再 push 就可以了。



## 2. 放弃自己的内容，维持远程的修改

类似的选择"Resolve Conflicts" -> "Resolve Using Theirs". 完成后的情况小有不同，本地文件内容已经被修改（被远程修改覆盖），需要再次commit然后push。



## 3. 根据自己的内容和远程的内容，给出一个和两者都不一样的内容，通常是包含两者的改动

最简单的方式就是自己直接打开文件，修改内容为自己需要的，比如：

```
aaaaaaaaaa10 - changed by myself and others
aaaaaaaaaa20
aaaaaaaaaa30
aaaaaaaaaa40
aaaaaaaaaa50
```

记得务必将三行标识符号删除

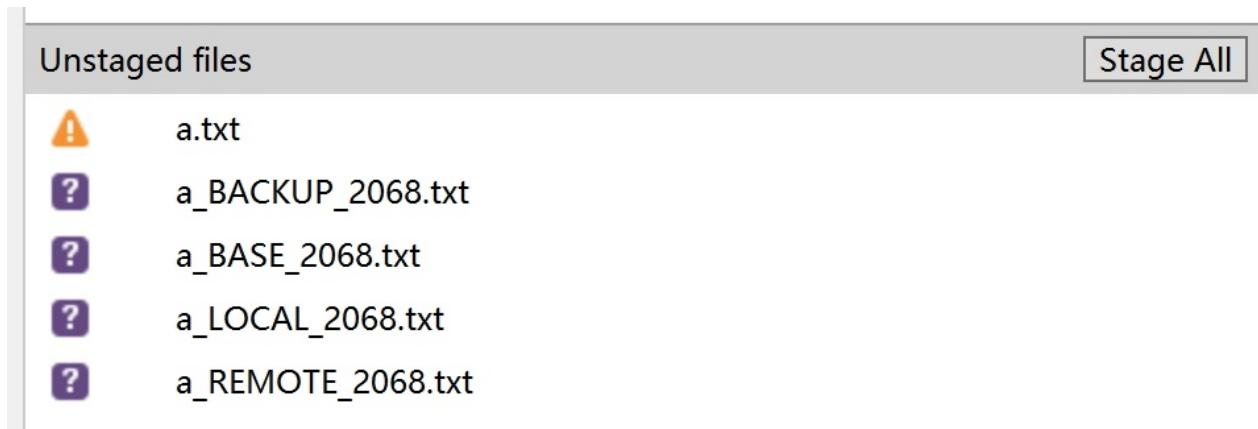
对于文件简单而冲突情况非常清晰的场景，这种方案可以简单的解决问题，但是当文件内容比较多，冲突情况复杂而不明朗时，尤其是有多个冲突，每个冲突点的取舍各不相同，需要逐个排查时，就需要有更好的方案了。

# 外部 Merge 工具

## 系统默认下的行为

默认情况下 sourcetree 是没有配置 external merge tool 的，即 "Tools" -> "Options" -> "Diff" 下面的 external merge tool 设置为 "System Default"

在发生 merge 冲突时，选 "Resolve Conflicts" -> "Launch External Merge Tool"，确认之后得到的 merge 结果如下：

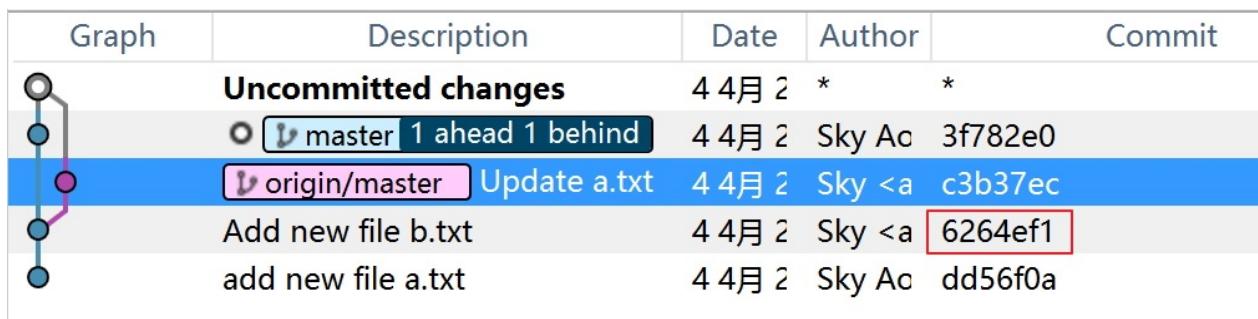


这里生成了多个文件(每个文件后缀中的数字,如这里的2068,可以不理会,应该只是为了避免重名)：

- a\_BACKUP\_####.txt : 自动 merge 完成之后的备份文件，里面的内容和 a.txt 一致
- a\_LOCAL\_####.txt : merge 操作中该文件的本地版本
- a\_REMOTE\_####.txt : merge 操作中该文件的远程版本
- a\_BASE\_####.txt : merge 操作中该文件的本地版本和远程版本的基线版本

## BASE 的概念

这里重点解释一下 BASE 的概念，下图是我们正在进行的 merge (由 pull 操作触发)：



图上可以看到两次的 commit，正是这两次 commit 修改了同一个文件的同一行，造成了我们现在面临的代码冲突。我们来看两次提交的信息，注意红框的内容：

```

Sorted by file status ▾ ☰
Commit: 3f782e0094df5abae6f04055be4fc14513767c36 [3f782e0]
Parents: 6264ef115a
Author: Sky Ao <aoxiaojian@gmail.com>
Date: 2017年4月4日 18:54:36
Labels: HEAD, -, master

update first line of a.txt

a.txt

Sorted by file status ▾ ☰
Commit: c3b37ec3419529c0fe7c37e524caf68f45a1da83 [c3b37ec]
Parents: 6264ef115a
Author: Sky <aoxiaojian@hotmail.com>
Date: 2017年4月4日 18:52:01
Labels: origin/master

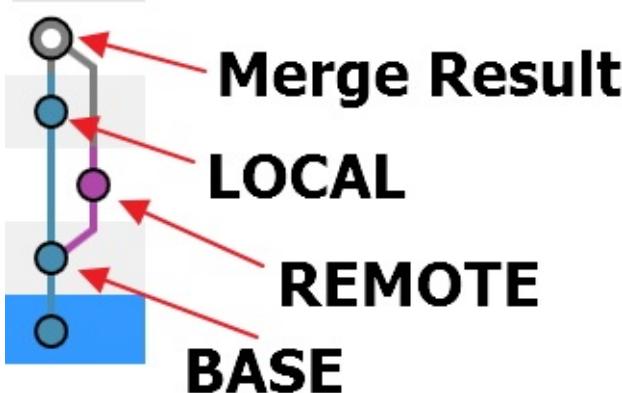
Update a.txt by others

a.txt

```

我们会发现这两次提交的 parent 都是 "6264ef115a"，这点从图形上也可以非常明显的体现出来，在 "6264ef115a" 这次提交之后，图形发生了一个分叉。然后我们现在又试图将这个分叉合并回来。

下图是图形上各个 commit 所表示的圆点和 merge 操作时的 LOCAL/REMOTE/BASE 概念的对应关系：



在 merge 的时候，一个至关重要的事情是：我们需要了解到底 merge 的两边分别做了什么改动，然后我们才能决定最后需要保留的内容。

这点体现在 merge tool 的工作原理上：

- LOCAL 和 REMOTE 可以从 merge 操作的两个分支上直接获取到，非常简单
- BASE 就需要推导，从 LOCAL 和 REMOTE 开始向前推，检查parent，直到两边的 parent 指向同一个 commit，这就得到了 BASE
- 然后， LOCAL 和 BASE 做一次 diff 操作，就知道 LOCAL 这边做了什么改动；同理

REMOTE那边做一次 diff

- 将两次 diff 的结果呈现给用户，以便用户判断和选择

## 使用 external merge tool

我们继续前面的话题，在没有配置 external merge tool 时，sourcetree 只能简单的生成各个文件（BASE/LOCAL/REMOTE等），而无法给出一个友好的图形界面。

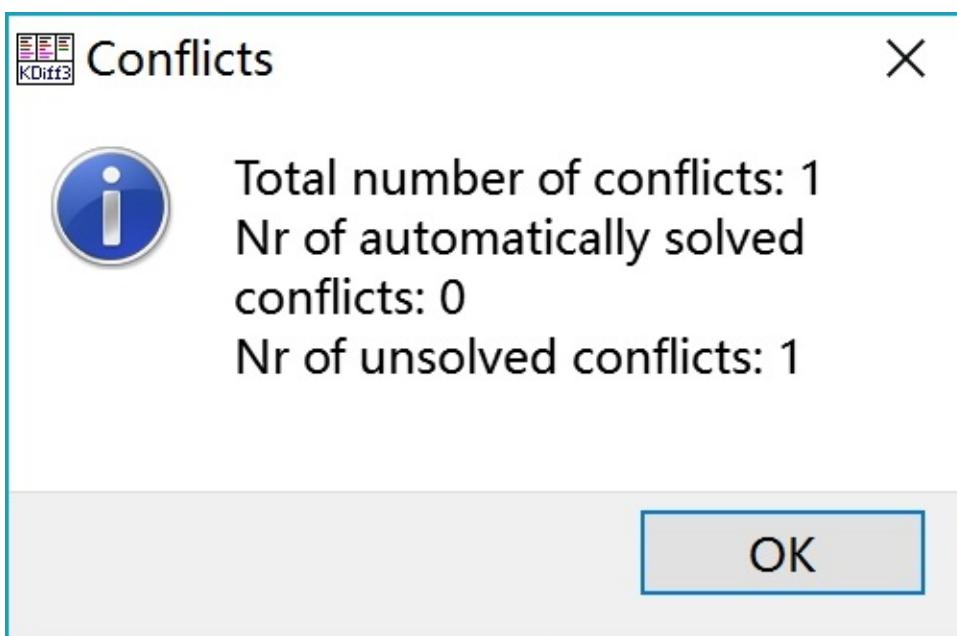
我们现在开始介绍如何使用 external merge tool

### kdiff3

再使用前请先安装 kdiff3，详细介绍见前面的 "设置" 一节。

然后修改配置，"Tools" -> "Options" -> "Diff" , external merge tool 设置为 "KDiff3".

我们再次进行merge，选 "Resolve Conflicts" -> "Launch External Merge Tool"，sourcetree 会调用kdiff3，然后提示如下：

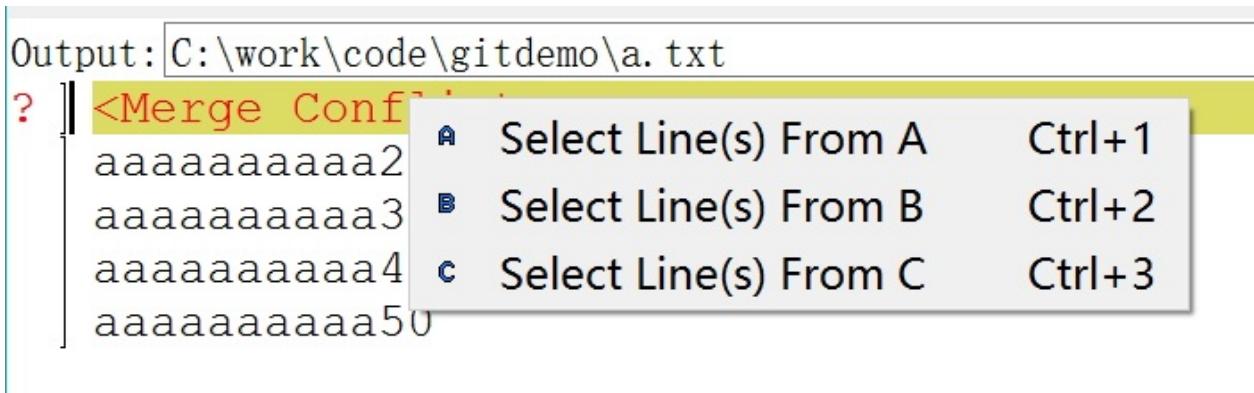


Kdiff 以这样的图形界面来展示文件冲突的情况：

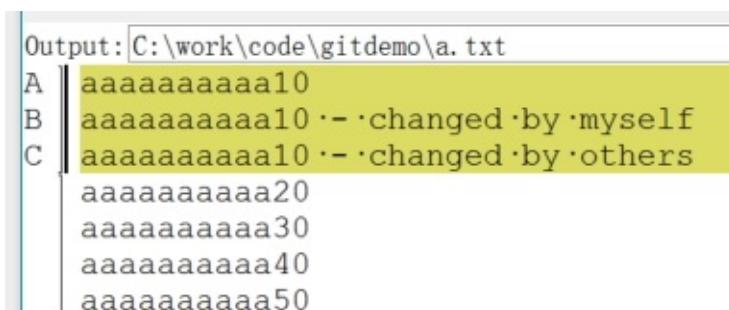
The KDiff3 interface shows three panes for files A (Base), B (Local), and C (Remote). The A and B panes show identical content: 'aaaaaaa10', 'aaaaaaaaaa20', 'aaaaaaaaaa30', 'aaaaaaaaaa40', and 'aaaaaaaaaa50'. The C pane shows the same lines with changes: 'aaaaaaa10' is highlighted in yellow ('changed by myself'), while the rest of the lines are greyed out. The bottom pane shows the merged output: 'aaaaaaa10' followed by a red conflict marker '<Merge Conflict>' and then the lines 'aaaaaaaaaa20', 'aaaaaaaaaa30', 'aaaaaaaaaa40', and 'aaaaaaaaaa50'.

- 第一排的三个窗口分别是 BASE / LOCAL / REMOTE
- 第二排的窗口显示的是当前 merge 的结果，其中发生冲突的地方显示为 <Merge Conflict>

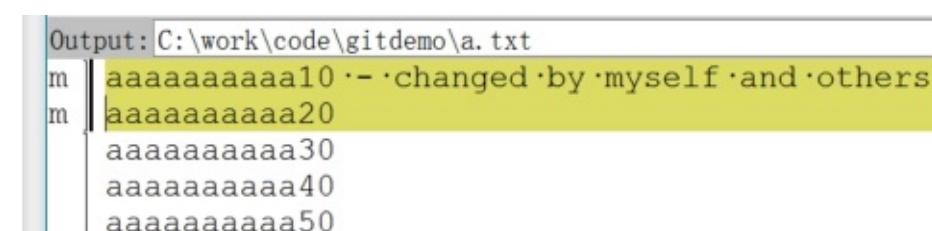
右键点冲突点所在的 <Merge Conflict>，弹出选择框：



在这里可以方便的选择希望 merge 后保留的内容。特别提醒，可以选择ABC中的一个或者多个！如图我选择了A/B/C三个：



也可以在这个基础上自行修改，比如我修改为保留BC两者的合集：



保存，退出kdiff3，回到本地文件系统，看到 merge 的结果：

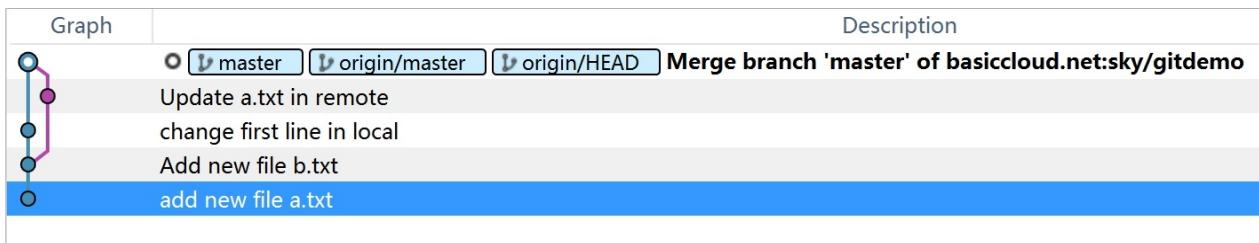
名称	修改日期	类型
<input checked="" type="checkbox"/> a.txt	2017/4/5 15:37	文本文档
<input type="checkbox"/> a.txt.orig	2017/4/5 11:10	ORIG 文件
<input type="checkbox"/> b.txt	2017/4/4 18:13	文本文档

好消息是原来的 BASE/LOCAL/REMOTE/BACKUP 文件都被清理掉了，工作区现在干净了很多。坏消息是还是生成了一个 \*.orig

注: 为了避免将.orig 文件提交到git仓库, 请修改 .gitignore 文件, 增加一行, 内容为  
\*.orig 或者干脆就不生成这个备份文件: git config --global mergetool.keepBackup  
false

## commit并push

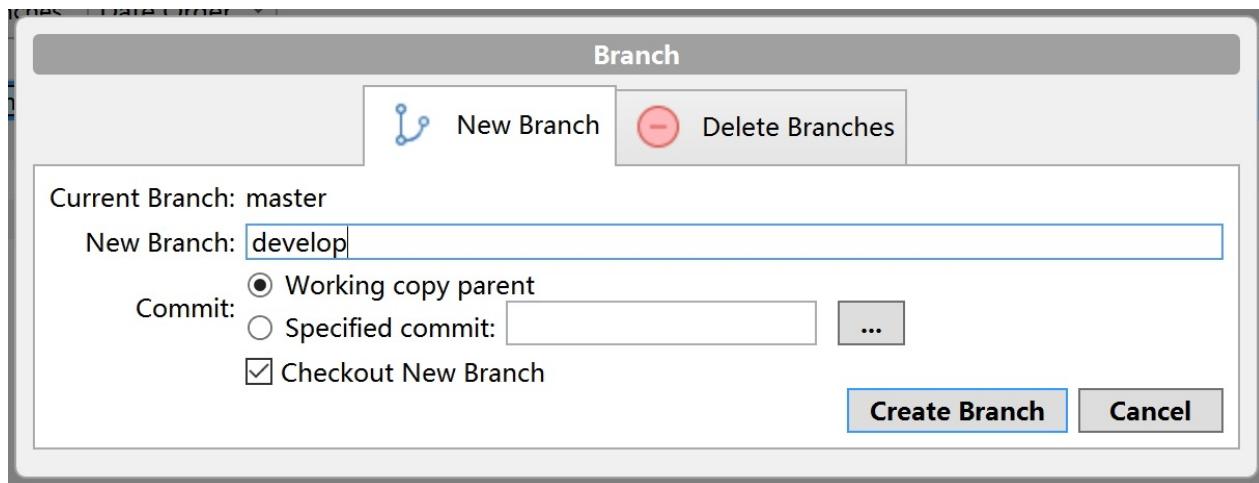
完成冲突的处理之后, 就可以再次commit然后push到远程仓库了, 此时 sourcetree 的显示如下:



# merge 时冲突

## 准备工作

我们先从 master 分支拉出一个 develop 分支



然后我们修改 a.txt 文件，内容如下：

```
aaaaaaaaaa10 - changed by myself and others
aaaaaaaaaa20
aaaaaaaaaa30 - changed in develop branch
aaaaaaaaaa50
aaaaaaaaaa60
```

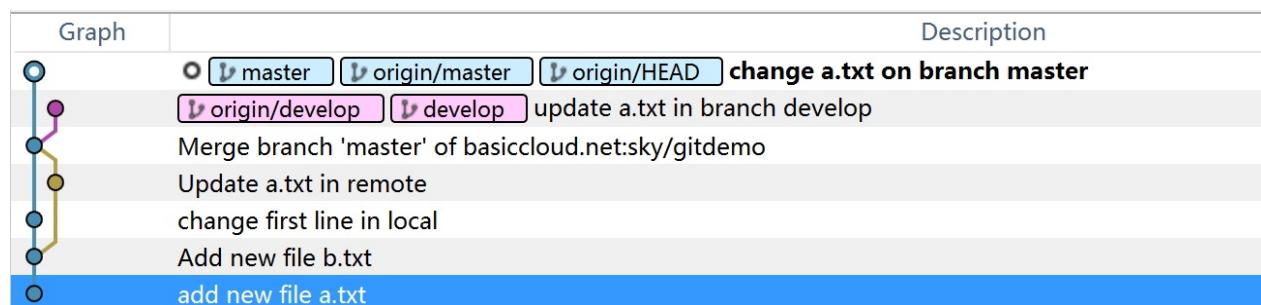
内容修改如下

- 修改了第三行的内容
- 删除了原来的第四行
- 在最后增加了一行

提交并 push 到远程仓库，然后 checkout 回 master branch，继续修改 a.txt 的内容：

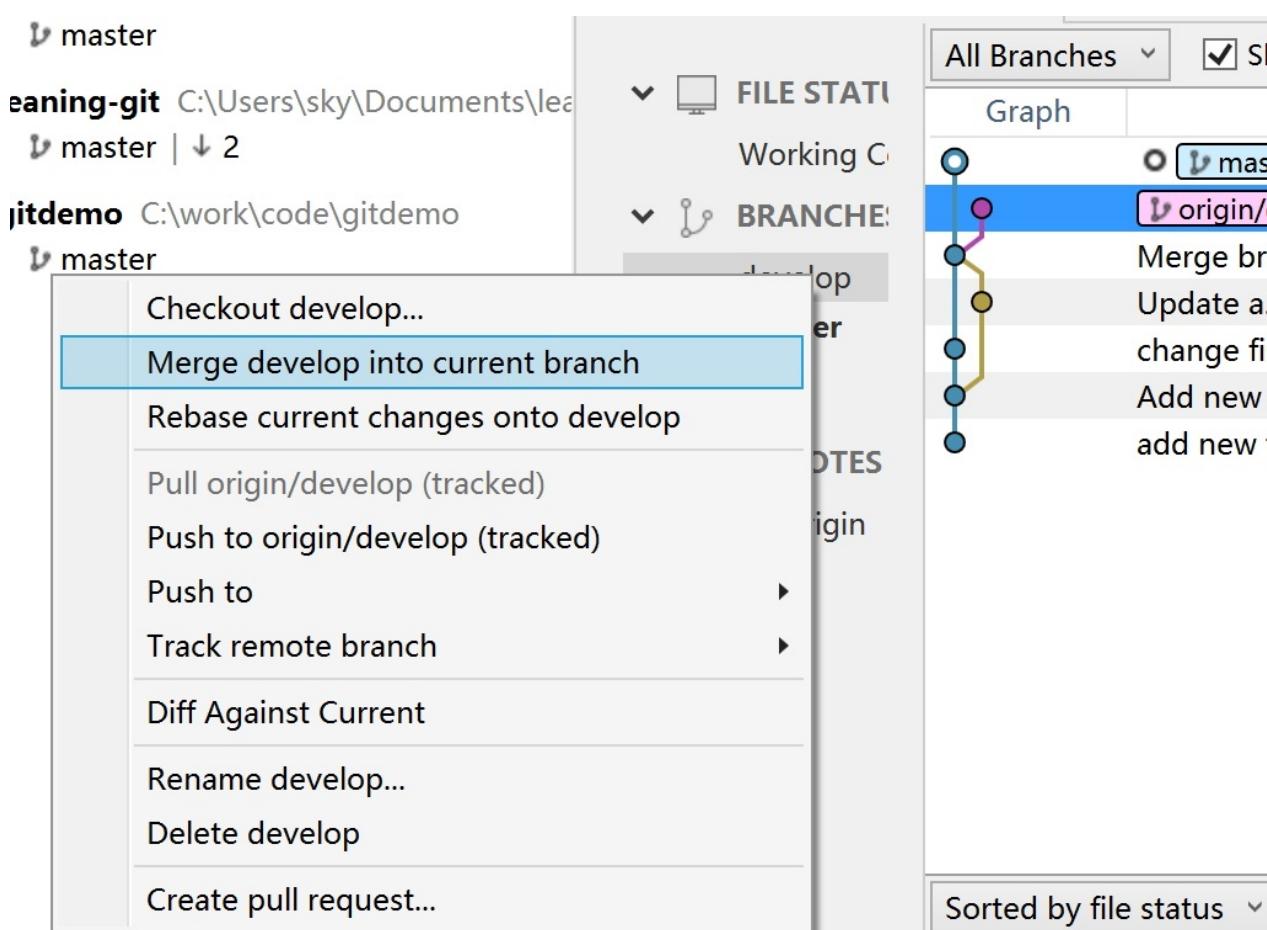
```
aaaaaaaaaa10 - changed by myself and others
aaaaaaaaaa20
aaaaaaaaaa30 - changed in master branch
aaaaaaaaaa40
aaaaaaaaaa50
```

这里我们故意修改第三行，以便造成文件冲突。



## 开始 merge

保持当前之分为 master，然后右键点 develop 分支，选择 "Merge develop into current branch":



提示冲突：

## Merge Conflicts

X



You now have merge conflicts in your working copy that need to be resolved before continuing.

You can do this by selecting the conflicted files and using the options under the 'Resolve Conflicts' menu.



Don't ask me again

关闭(C)

此时 a.txt 文件的内容如下，包含 merge 冲突内容：

```
aaaaaaaaaa10 - changed by myself and others
aaaaaaaaaa20
<<<<< HEAD
aaaaaaaaaa30 - changed in master branch
aaaaaaaaaa40
aaaaaaaaaa50
=====
aaaaaaaaaa30 - changed in develop branch
aaaaaaaaaa50
aaaaaaaaaa60
>>>>> develop
```

后面的操作和 pull 时重复的处理一致。