Git 教案

1-1 主要内容

- 什么是 Git
- Git 安装配置
- Git 工作流程
- Git 工作区、暂存区和版本库
- 创建仓库
- 基本操作
- 分支管理
- 查看提交历史
- 标签
- 远程仓库
- 服务器搭建
- 开发工具集成

1-2 课程内容

1-2.1 什么是 Git

Git 是一个开源的分布式版本控制系统，用于敏捷高效的处理任何项目的版本问题。
Git 是 Linus Torvalds 为了帮助管理 Linux 内核开发而开发的一个开放源码的版本控制软件。
Git 与常用的版本控制工具 CVS,SVN 等不同，它采用了分布式版本库的方式，不必服务器端软件支持。

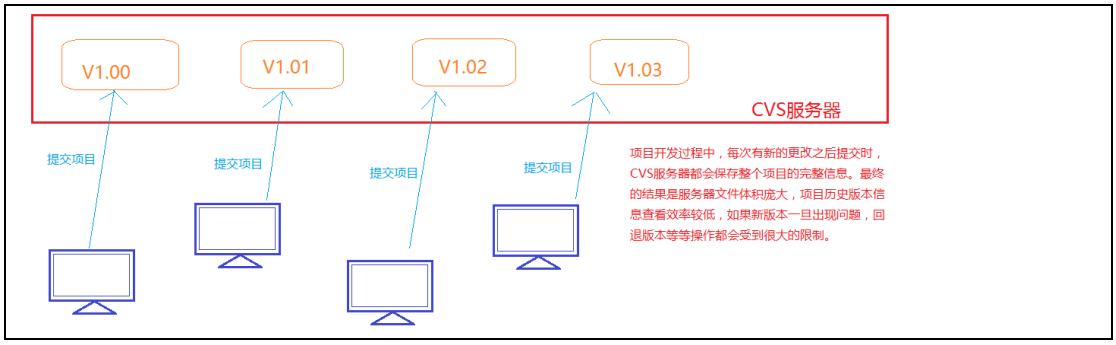
1-2.2 关于项目版本管理

项目在开发的过程中，经常会出现多人分工协作进行项目分发并开发整合的过程，所以项目在刚开始流行的时候经常会出现一些协作开发的同步的问题，同时存在项目整体进度的控制和管理的问题，所以在程序开发行业衍生出来了版本管理工具

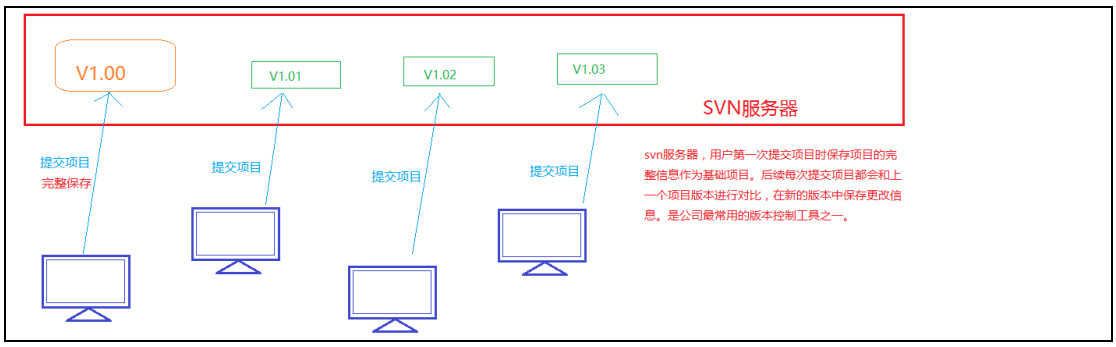
版本管理工具，首先是一个内容管理工具，可以将项目的内容信息存放在版本管理服务器上方便项目组人员进行访问和查询修改。

版本管理具有里程碑意义的主要有三个阶段
CVS 阶段→ SVN 阶段 → Git 阶段

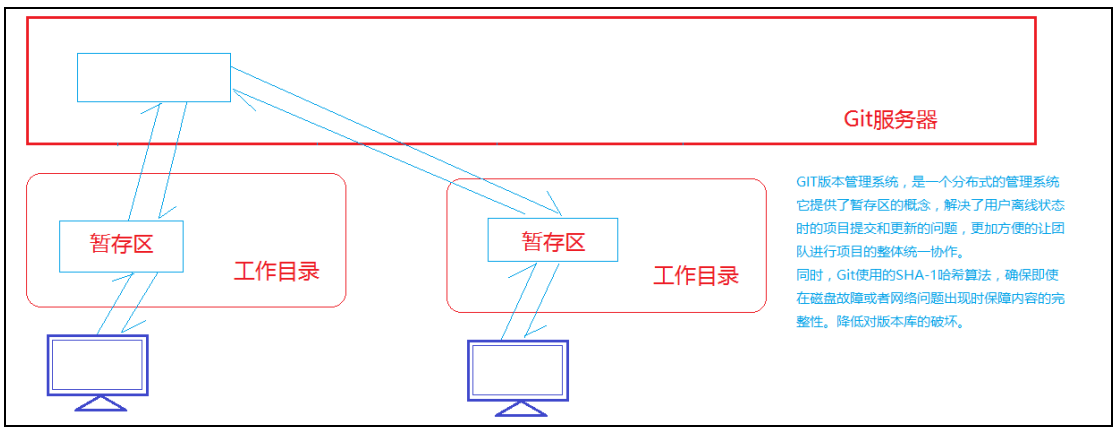
- CVS 阶段
  - 项目搭建开发过程中，每次提交项目都会将整个项目提交到服务器进行保存，服务器存储着项目的 N 个备份，开发过程中的协作效率较低，同时也出现了各种传输的问题，所以慢慢淡出了行业。



- SVN 阶段
  - 考虑到 CVS 的缺陷，开发人员根据项目的实际情况，研发出专门针对项目版本控制的软件 Subversion（简称 SVN），SVN 同样也是搭建服务器，让项目组成员将数据存储在服务器上，但是每次改动并提交的时候，SVN 服务器并不重新保存整个项目的完整信息，而是和原来的项目进行对比，只保存改动的信息。这样就在很大的程度上对于项目版本服务器、项目协作效率有了显著的提升。所以至今为止，有很多公司依然选用 SVN 作为公司内部项目协作的版本控制软件



- Git 阶段
  - 前面的 CVS 和 SVN 都是基于一个服务器的，如果脱离服务器，项目的版本保存就没有了任何意义，Git 恰恰处理了这样的问题，Git 是一个分布式的版本控制系统，在 Git 中即使用户离线，也能进行项目的提交和更新操作，等到下次连线服务器时进行整体的同步操作。



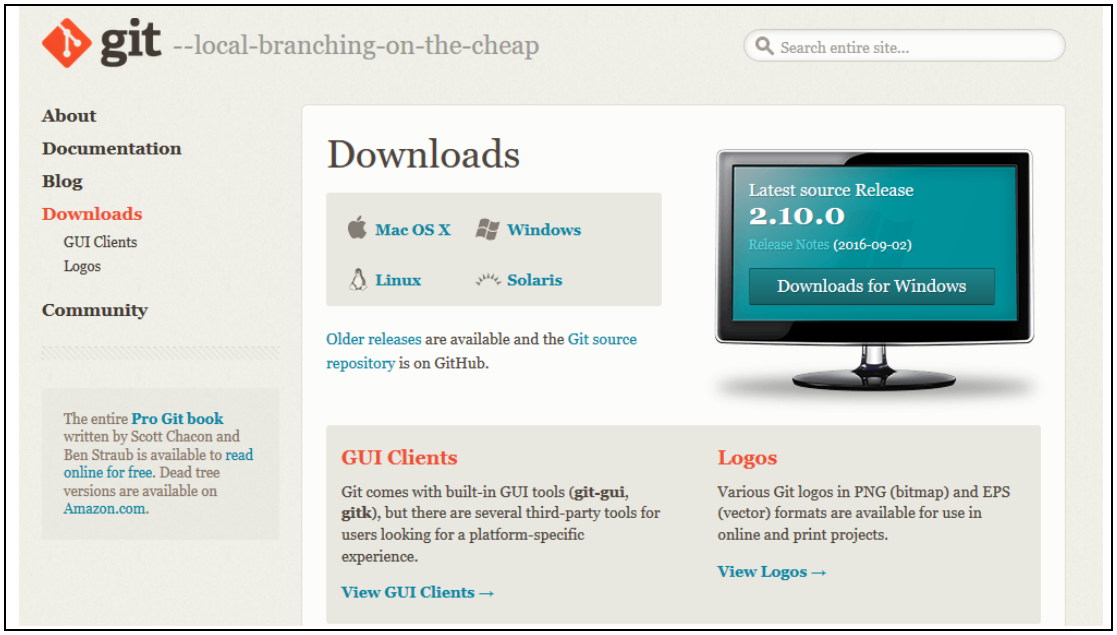
1-2.3 Git VS SVN

- git 是分布式的，svn 不是
- git 按照元数据的方式存储内容，svn 是按照文件的形式存储
- git 和 svn 中的分支不同
- git 没有全局版本号，svn 有
- git 内容的完整性优于 svn

1-2.4 安装配置

使用 git 之前，PC 上需要安装 Git
支持 Linux/Unix、Solaris、Mac 和 Windows 平台上运行

Git 各平台安装包下载地址为：http://git-scm.com/downloads



选中对应的操作系统平台，下载对应的软件安装包。

1-2.5 git 配置

git 提供了一个 git config 工具，专门用于配置和读取相应的工作环境变量

- etc/gitconfig 文件，系统中所有用户都普遍适用的配置，如果适用 get config 时添加--system 选项，修改的就是这个文件
- ~/.gitconfig 文件，用户目录下的配置文件，只适用于当前用户，使用 get config 时添加--global 选项，修改的就是这个文件

- /config 当前项目的.git 目录中的配置文件，配置只是针对当前项目有效。

每一个级别中的配置都会覆盖上一个级别的配置，所以.git/config 中的配置会覆盖 etc/gitconfig 中的配置信息。

- 配置案例：
  - 配置个人用户信息和电子邮件地址

```
FLEX@FLEX-MU MINGW64 /d/RESOURCE/GIT_REPO (master)
$ git config --global user.name "muwenbin"

FLEX@FLEX-MU MINGW64 /d/RESOURCE/GIT_REPO (master)
$ git config --global user.email "1007821300@qq.com"
```

- 查看配置信息
  - 使用 get config --list 命令查看

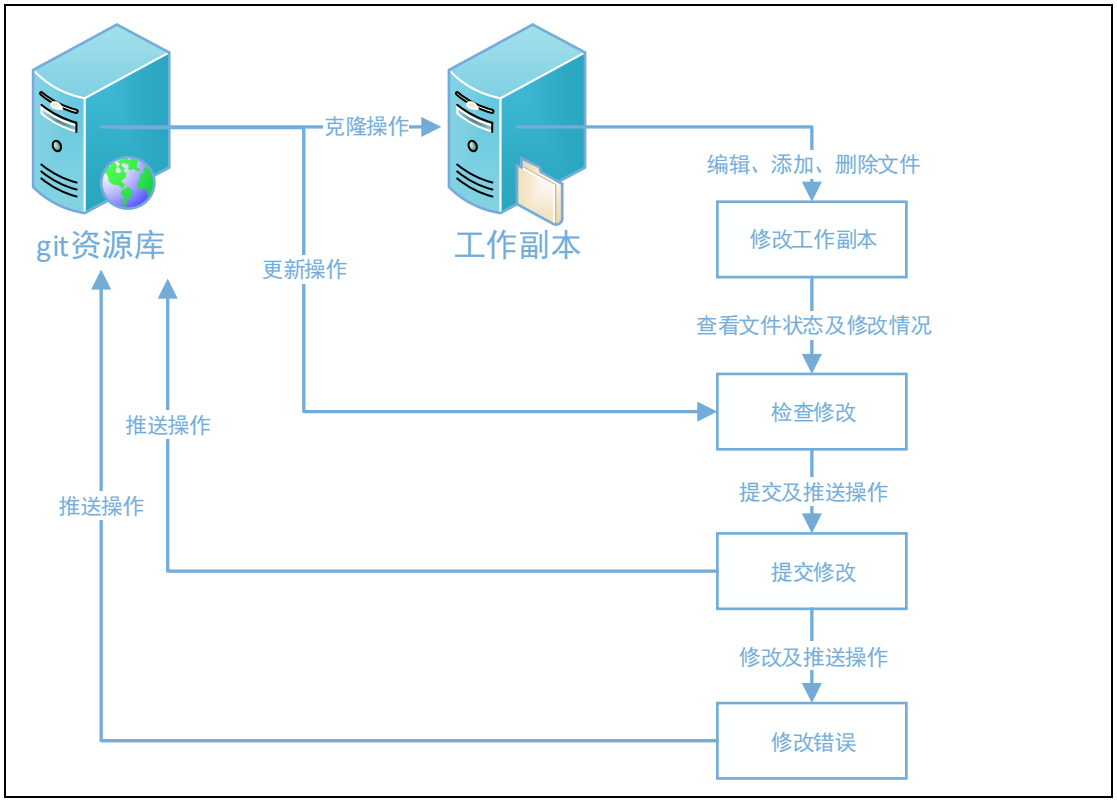
```
FLEX@FLEX-MU MINGW64 /d/RESOURCE/GIT_REPO (master)
$ git config --list
core.symlinks=false
core.autocrlf=true
core.fscache=true
color.diff=auto
color.status=auto
color.branch=auto
color.interactive=true
help.format=html
http.sslcainfo=C:/Program Files/Git/mingw64/ssl/certs/ca-bundle.crt
diff.astextplain.textconv=astextplain
rebase.autosquash=true
credential.helper=manager
user.name=muwenbin
user.email=1007821300@qq.com
core.repositoryformatversion=0
core.filemode=false
core.bare=false
core.logallrefupdates=true
core.symlinks=false
core.ignorecase=true
```

1-2.6 Git 工作流程

常规工作流程如下：

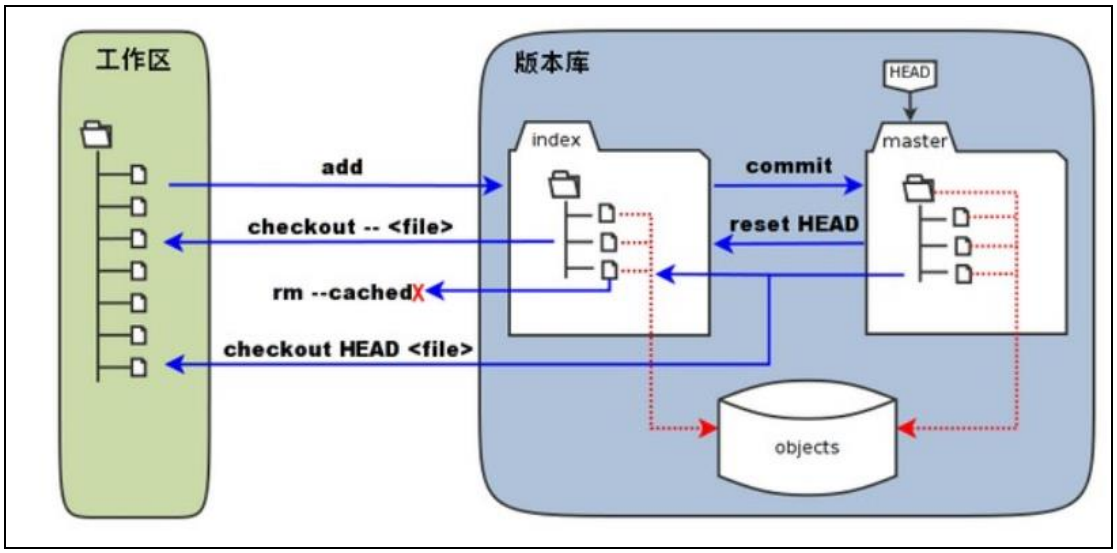
- 克隆 Git 资源作为目录
- 在克隆的资源上添加或者修改文件
- 如果是其他人修改过的文件，可以进行更新文件的操作
- 提交前的文件修改信息检查
- 提交修改
- 修改完成后，如果发现错误，撤回提交并再次修改后提交

实际操作过程如下图



1-2.7 工作区、暂存区及版本库

- 基本概念
  - 工作区：个人 PC 中能看到的文件目录结构
  - 暂存区：stage/index，一般存放在.git/index 中，所以 git 中的暂存区也会称为索引
  - 版本库：工作区中的.git 隐藏文件，不算是工作区，而是 Git 的版本库。



图中左侧为工作区，右侧为版本库。在版本库中标记为 "index" 的区域是暂存区（stage, index），标记为 "master" 的是 master 分支所代表的目录树。

图中我们可以看出此时 "HEAD" 实际是指向 master 分支的一个"游标"。所以图示的命令中出现 HEAD 的地方可以用 master 来替换。
图中的 objects 标识的区域为 Git 的对象库，实际位于 ".git/objects" 目录下，里面包含了创建的各种对象及内容。
当对工作区修改（或新增）的文件执行 "git add" 命令时，暂存区的目录树被更新，同时工作区修改（或新增）的文件内容被写入到对象库中的一个新的对象中，而该对象的 ID 被记录在暂存区的文件索引中。
当执行提交操作（git commit）时，暂存区的目录树写到版本库（对象库）中，master 分支会做相应的更新。即 master 指向的目录树就是提交时暂存区的目录树。
当执行 "git reset HEAD" 命令时，暂存区的目录树会被重写，被 master 分支指向的目录树所替换，但是工作区不受影响。
当执行 "git rm --cached <file>" 命令时，会直接从暂存区删除文件，工作区则不做出改变。
当执行 "git checkout ." 或者 "git checkout -- <file>" 命令时，会用暂存区全部或指定的文件替换工作区的文件。这个操作很危险，会清除工作区中未添加到暂存区的改动。
当执行 "git checkout HEAD ." 或者 "git checkout HEAD <file>" 命令时，会用 HEAD 指向的 master 分支中的全部或者部分文件替换暂存区和以及工作区中的文件。这个命令也是极具危险性的，因为不但会清除工作区中未提交的改动，也会清除暂存区中未提交的改 动。

1-2.8 创建仓库

- git init

Git 使用 git init 命令初始化一个 git 仓库。
git 很多命令都是在仓库中运行的，git init 执行完成后 Git 会在对应的目录中创建一个.git 隐藏文件夹，文件夹中的文件包含了 git 所有元数据，其他的项目文件夹保持不变

- 使用当前文件夹作为 git 仓库
  - 进入对应的文件夹中，执行 git init 命令即可

- 指定文件夹作为 git 仓库
  - 执行 git init myrepo;
  - 执行完成后，myrepo/文件夹下会出现一个.git 目录，

创建好的项目文件夹中，就可以通过 git add file_name 命令，就可以告诉 git 对指定的文件进行跟踪操作。

```
MINGW64:/d/RESOURCE/GIT_REPO
FLEX@FLEX-MU MINGW64 /d/RESOURCE/GIT_REPO (master)
$ git add file1

FLEX@FLEX-MU MINGW64 /d/RESOURCE/GIT_REPO (master)
$ git add file2
```

最后对修改的文件进行提交操作

```
FLEX@FLEX-MU MINGW64 /d/RESOURCE/GIT_REPO (master)
$ git commit -m "初始化项目版本"
[master (root-commit) 6d230dd] 初始化项目版本
 3 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 README
 create mode 100644 file1
 create mode 100644 file2
```

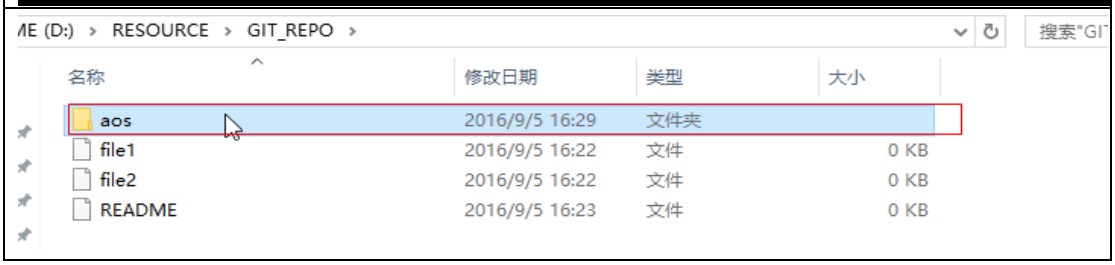
- 克隆项目

所谓克隆项目，就是创建远程 git 仓库中的项目副本

- git clone <repo>
- git clone <repo> <directory>

案例操作

```
FLEX@FLEX-MU MINGW64 /d/RESOURCE/GIT_REPO (master)
$ git clone https://github.com/michalsnik/aos.git
Cloning into 'aos'...
remote: Counting objects: 1014, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 1014 (delta 0), reused 0 (delta 0), pack-reused 1011
Receiving objects: 100% (1014/1014), 992.81 KiB | 186.00 KiB/s, done.
Resolving deltas: 100% (499/499), done.
```



1-2.9 基本操作【核心】

- 获取与创建项目命令
  - git init [<directoryName>]：初始化创建项目工作空间
  - git clone <gitURL> [projectName]：克隆项目

```
FLEX@FLEX-MU MINGW64 /d/RESOURCE/GIT_REPO (master)
$ git clone https://github.com/michalsnik/aos.git
Cloning into 'aos'...
remote: Counting objects: 1014, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 1014 (delta 0), reused 0 (delta 0), pack-reused 1011
Receiving objects: 100% (1014/1014), 992.81 KiB | 186.00 KiB/s, done.
Resolving deltas: 100% (499/499), done.
```

- 添加文件到缓存中
  - git add <fileName>

```
FLEX@FLEX-MU MINGW64 /d/RESOURCE/GIT_REPO (master)
$ git add README
```

- 查看上次提交之后的修改状态
  - git status

```
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   README
```

- 查看具体的修改信息
  - git diff：查看尚未缓存的改动{尚未 add 操作}
  - git diff --cached：查看已经缓存的改动
  - git diff HEAD:查看已经缓存和未缓存的所有改动
  - git diff --stat:查看摘要信息

```
$ git diff
diff --git a/README b/README
index e69de29..1ee195c 100644
--- a/README
+++ b/README
@@ -0,0 +1 @@
+测试案例
\ No newline at end of file
```

- 添加内容到仓库中
  - git commit:将缓存区中的内容添加到仓库中。
  - git 为你的每一个提交记录名字和电子邮箱地址【教程开头使用 git config 配置的用户信息】
  - git commit -m “注释内容”：-m 选项用于提交时添加注释内容

```
FLEX@FLEX-MU MINGW64 /d/RESOURCE/GIT_REPO (master)
$ git commit -m "提交所有修改内容"
[master 1edb7a8] 提交所有修改内容
 1 file changed, 1 insertion(+)
```

  - 同时 git 提供了-a 选项，用于提示提交简略信息

```
FLEX@FLEX-MU MINGW64 /d/RESOURCE/GIT_REPO (master)
$ git commit -am "用户须知修改"
[master c25e6c1] 用户须知修改
 1 file changed, 1 insertion(+), 1 deletion(-)
```

- 取消缓存内容
  - git reset HEAD:用于取消用户已经缓存的内容

修改 README 和 file2 文件

```
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   README
        modified:   file2

no changes added to commit (use "git add" and/or "git commit -a")
```

取消 file2 缓存

`$ git reset HEAD -- file2`
`Unstaged changes after reset:`
`M  README`
`M  file2`

提交文件

- 从缓存区中移除
  - **git rm <fileName>**:将指定的文件从缓存区中移除
- 重命名并重新添加到缓存中
  - **git mv <fileName> <newFileName>**:将指定的文件 fileName 重新指定新的名称 newFileName,并自动通过 add 将新的文件添加到的缓存区中

```
FLEX@FLEX-MU MINGW64 /d/RESOURCE/GIT_REPO (master)
$ git mv README README.md

FLEX@FLEX-MU MINGW64 /d/RESOURCE/GIT_REPO (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        renamed:    README -> README.md
```

1-2.10 分支管理

很多的版本控制工具都是支持分支操作的,使用分支操作的意义是从开发的主线上分离开达到不影响主线任务的同时进行功能开发

创建分支使用如下命令

- git branch [branch_name]

切换分支命令

- git checkout [branck_name]

合并分支命令

- git merge

列出分支

- git branch

删除分支

- git -d [branch_name]

1-2.11 查看提交历史

使用 git 提交/更新等操作、或者克隆项目等操作后，很多情况下需要查看某个项目的操作历史记录信息

- git log:专门用于查看所有的 git 仓库操作历史数据的命令
- git log --online:查看项目的历史纪录的简洁版本
- git log --graph:查看历史记录中的分支、合并等操作
- git log --reverse:逆向显示所有的日志
- git log --author:查看指定用户的提交日志
- git log --before={3.week.age} --after{2015-10-11}:指定时间段查询
- git log --no-merges:隐藏合并提交

```
>> git log
```

```
$ git log
commit 75b98b48898557dadfeb75048f25025fc846d040
Author: muwenbin <1007821300@qq.com>
Date:   Mon Sep 5 17:02:04 2016 +0800

    取消缓存测试

commit c25e6c14e45c44a94b169110ce342e9a6ef0202a
Author: muwenbin <1007821300@qq.com>
Date:   Mon Sep 5 16:58:07 2016 +0800

    用户须知修改

.. .. .. .. .. .. ..

commit c25e6c14e45c44a94b169110ce342e9a6ef0202a
Author: muwenbin <1007821300@qq.com>
Date:   Mon Sep 5 16:58:07 2016 +0800

    用户须知修改

commit 1edb7a8ad089427e882c8f94c2d08d39eeef55ff
Author: muwenbin <1007821300@qq.com>
Date:   Mon Sep 5 16:56:13 2016 +0800

    提交所有修改内容

commit 219f09c785cd13302e439c3046f978c687776999
Author: muwenbin <1007821300@qq.com>
Date:   Mon Sep 5 16:49:02 2016 +0800
```

```
    test

commit 6d230dd15c8531c5d5a1ed10560e57d9cc5a6fa9
Author: muwenbin <1007821300@qq.com>
Date:   Mon Sep 5 16:23:51 2016 +0800

    初始化项目版本
```

1-2.12 标签操作

项目开发的过程中，会出现一些重要重大的改动，通常会称为里程碑版本等等，显示这个版本的重要性，在使用 Git 操作的时候，同样也可以通过特殊的方式标记某个版本的特殊性。

- `git tag <desc>`

给项目添加应标识/标签

- `git tag`

查看项目标签信息

```
FLEX@FLEX-MU MINGW64 /d/RESOURCE/GIT_REPO (master)
$ git tag
V1.00
V1.01
```

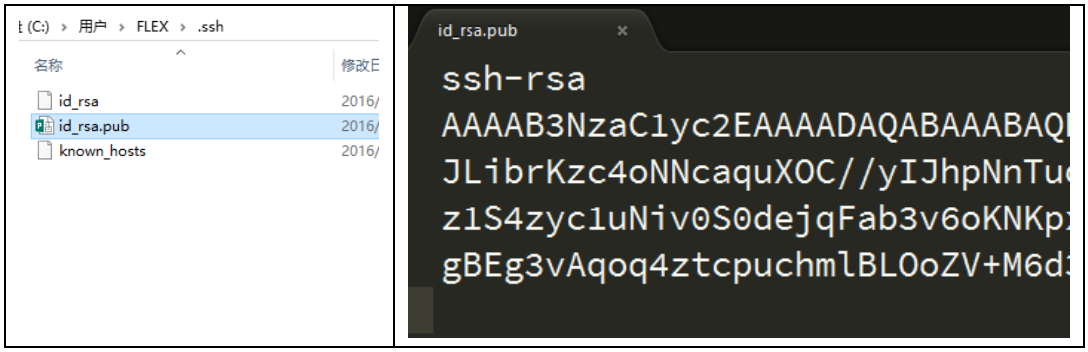
1-2.13 远程仓库

以上所有的操作都是在本地执行，但是如果要团队协作开发项目，或者将项目分享给其他人的话，就需要将项目数据放在其他人都能访问的服务器上。

通过使用下面的命令创建连接 SSH 密钥

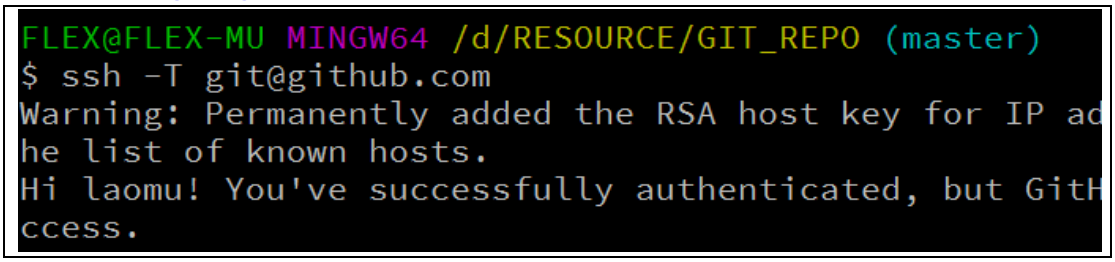
- `ssh-keygen -t rsa -C youemail@example.com`

命令会提示输入账号密码等之类的，直接回车。最终会在.ssh/目录中生成对应的 id_rsa.pub 密钥文件



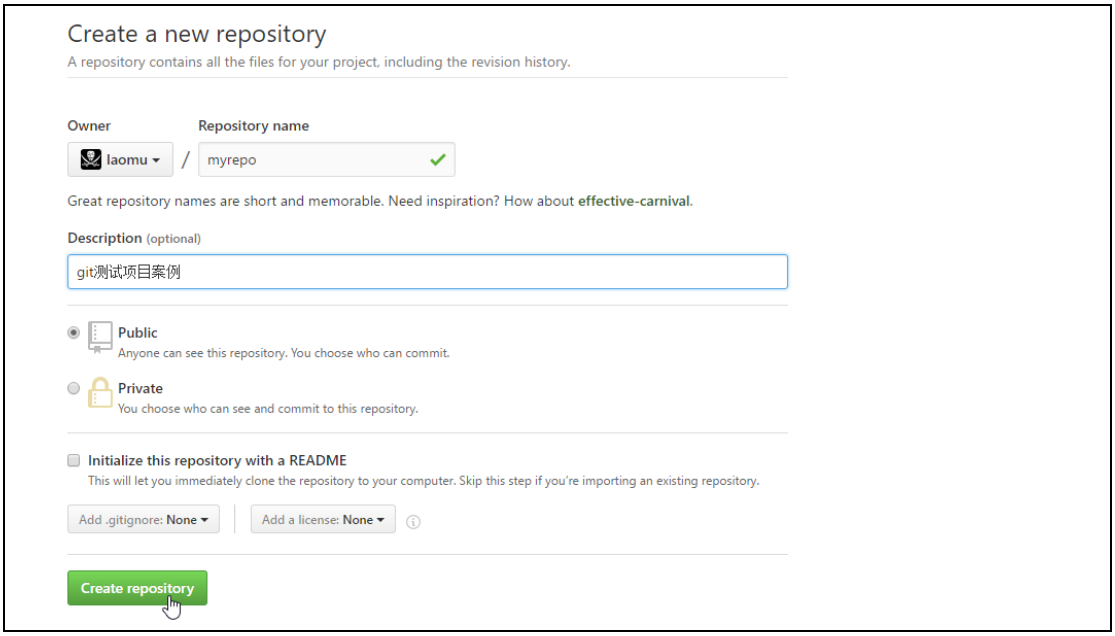
通过如下命令进行测试远程服务器连接是否成功

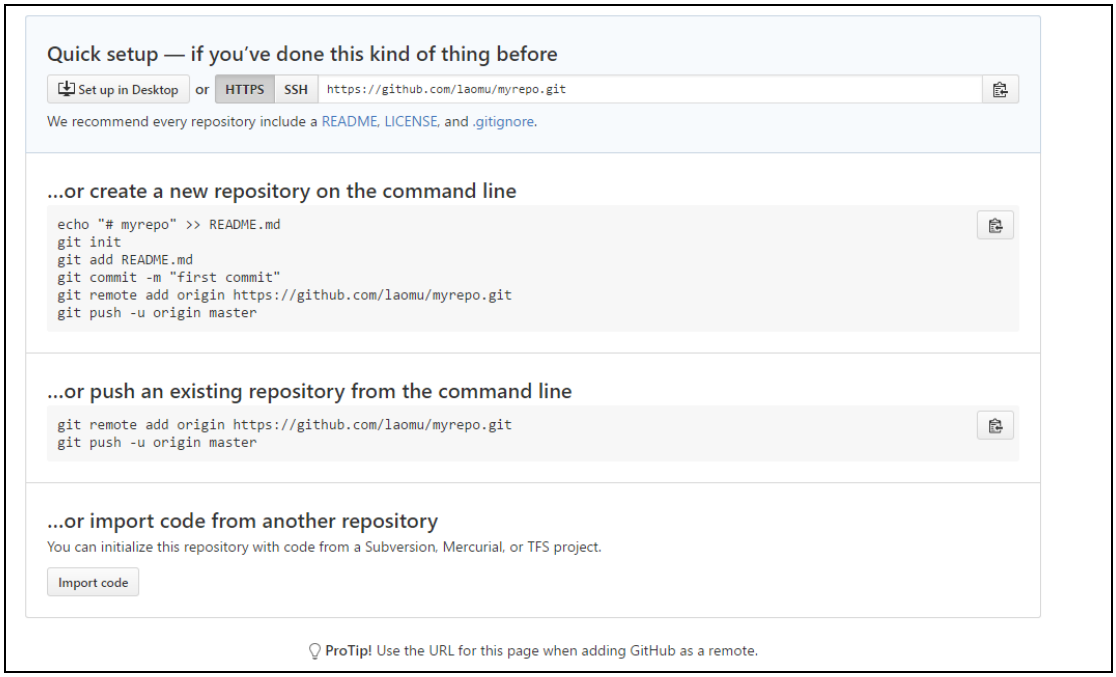
- ssh -T git@github.com



1-2.14 远程操作

登录 github.com 网站，点击 New repositoy 创建项目仓库。





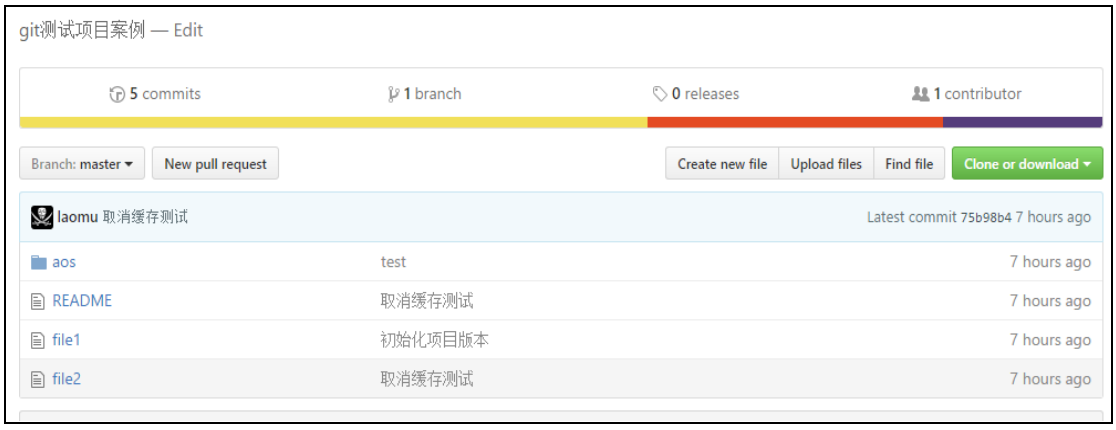
以上信息提示，可以从当前的仓库中克隆项目，也可以将本地项目推送到 GitHub 仓库中。

执行如下命令，将项目推送到远程仓库中

```
FLEX@FLEX-MU MINGW64 /d/RESOURCE/GIT_REPO (master)
$ git remote add origin http://github.com/laomu/myrepo.git

FLEX@FLEX-MU MINGW64 /d/RESOURCE/GIT_REPO (master)
$ git push -u origin master
Counting objects: 61, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (56/56), done.
Writing objects: 100% (61/61), 52.75 KiB | 0 bytes/s, done.
Total 61 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), done.
To http://github.com/laomu/myrepo.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.
```

远程仓库查看信息



1-2.15 远程操作命令

- 查看远程仓库
  - git remote
  - git remote -v
    - -v 参数可以查看到每个仓库别名的实际连接地址

```
FLEX@FLEX-MU MINGW64 /d/RESOURCE/GIT_REPO (master)
$ git remote
origin

FLEX@FLEX-MU MINGW64 /d/RESOURCE/GIT_REPO (master)
$ git remote -v
origin  http://github.com/laomu/myrepo.git (fetch)
origin  http://github.com/laomu/myrepo.git (push)
```

- 提取远程仓库
  - git fetch

```
FLEX@FLEX-MU MINGW64 /d/RESOURCE/GIT_REPO (master)
$ git fetch
```

备注：命令执行完成后，需要通过 git merge 合并远程分支到你自己的分支中

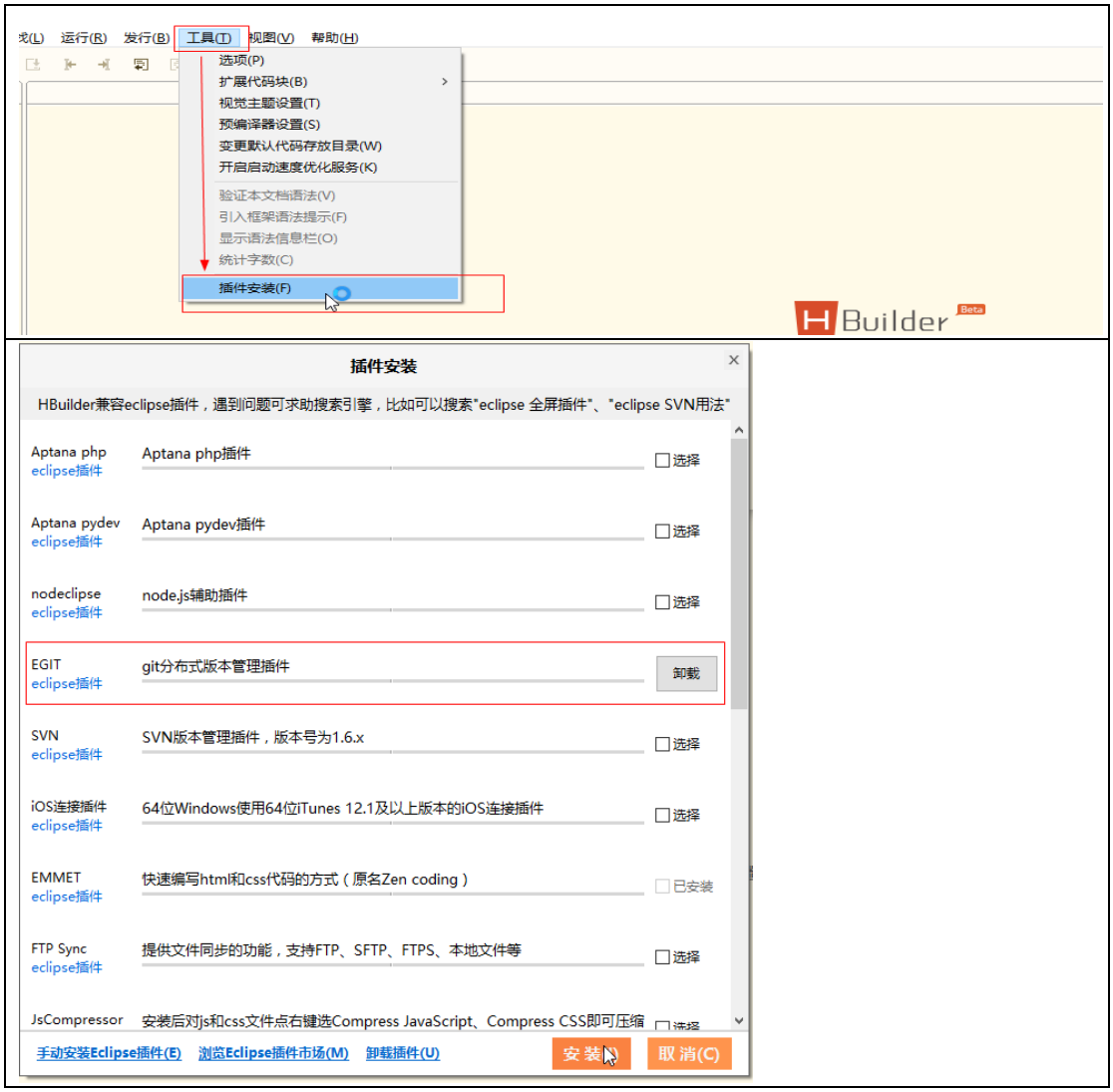
  - git pull:从远程仓库提取数据并尝试合并到当前分支

```
FLEX@FLEX-MU MINGW64 /d/RESOURCE/GIT_REPO (master)
$ git pull
Already up-to-date.
```

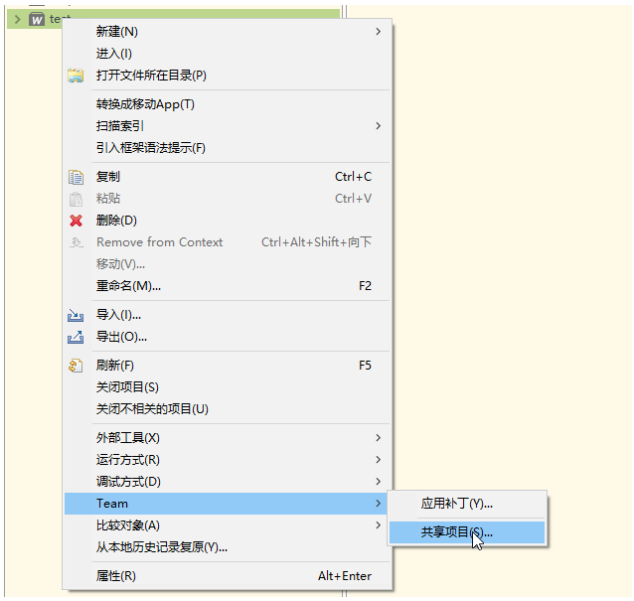
- 推送到远程仓库
  - git push [alias] [branch]
  - 将 branch 分支推送称为 alias 远程仓库的 branch 分支
- 删除远程仓库
  - git remote rm [repo_name]

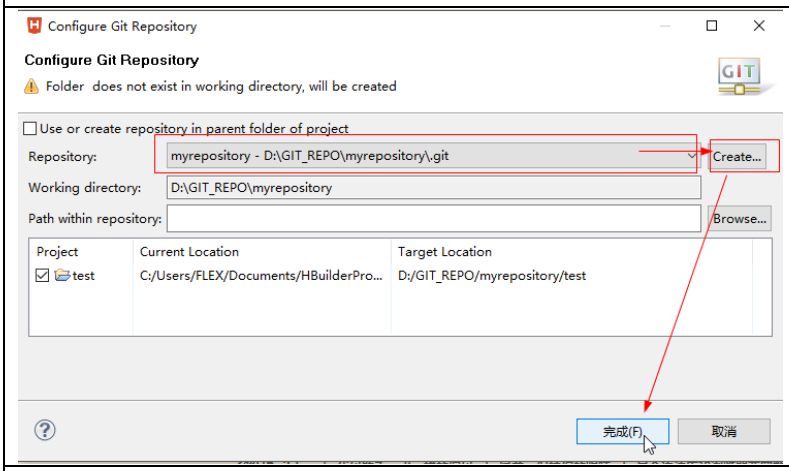
1-2.16 IDE 工具集成 GIT

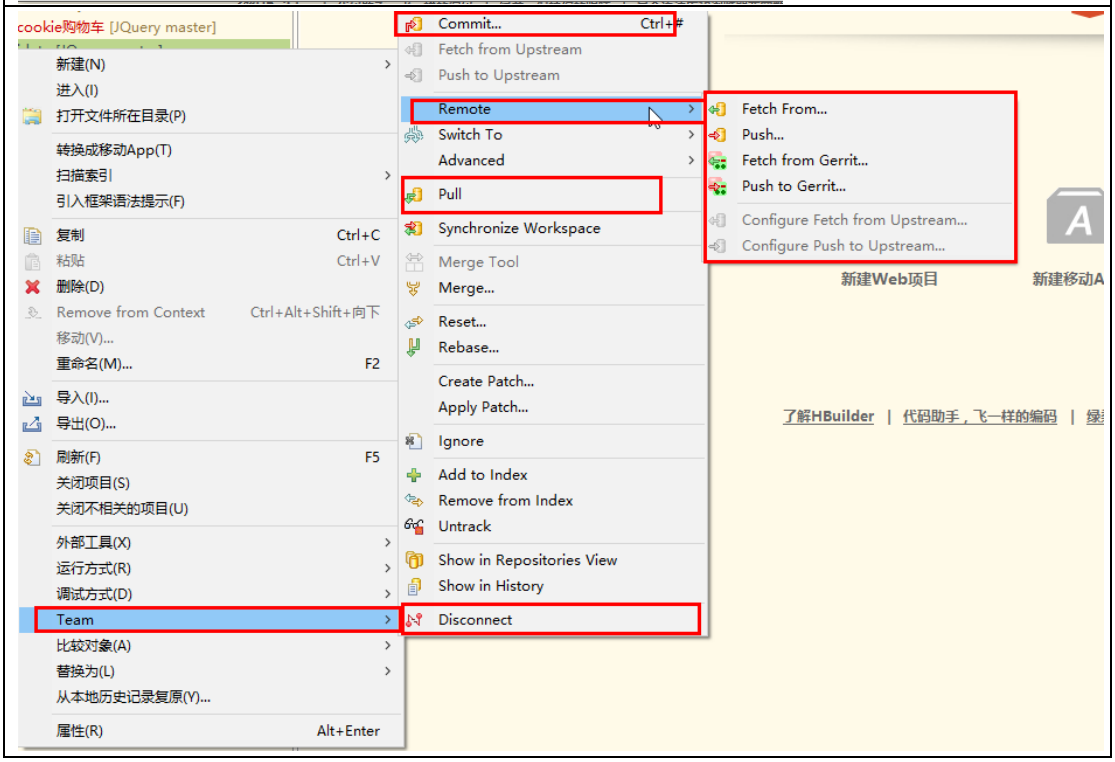
- HBuilder 集成 GIT 开发
  - 菜单-> 插件安装
  - EGIT->安装



  - 共享项目
    - 项目上右键点击
    - 选择 Team->共享项目
    - 在 Repository 中选择一个本地的 GIT 工作区
      - 工作区可以先选择一个目录文件夹
      - 点击 Create 按钮，会将选中的文件夹创建成 git 工作区
    - 点击完成，就会将要共享的项目剪贴到工作区中
      - 以后修改和提交的项目，都是 git 管理的项目，可以进行修改、提交、回退等等各种操作。


新建(N)
进入(I)
打开文件所在目录(P)
转换成移动App(T)
扫描索引
引入框架语法提示(F)
复制 Ctrl+C
粘贴 Ctrl+V
删除(D)
Remove from Context Ctrl+Alt+Shift+向下
移动(V)...
重命名(M)... F2
导入(I)...
导出(O)...
刷新(F) F5
关闭项目(S)
关闭不相关的项目(U)
外部工具(X)
运行方式(R)
调试方式(D)
Team
比较对象(A)
从本地历史记录复原(Y)...
属性(R) Alt+Enter
应用补丁(Y)...
共享项目(E)...


Configure Git Repository
Configure Git Repository
Folder does not exist in working directory, will be created
Use or create repository in parent folder of project
Repository:
myrepository - D:\GIT_REPO\myrepository\.git
Create...
Working directory:
D:\GIT_REPO\myrepository
Path within repository:
Browse...
Project
Current Location
Target Location
test
C:/Users/FLEX/Documents/HBuilderPro...
D:/GIT_REPO/myrepository/test
完成(F)
取消


cookie购物车 [JQuery master]
新建(N)
进入(I)
打开文件所在目录(P)
转换成移动App(T)
扫描索引
引入框架语法提示(F)
复制 Ctrl+C
粘贴 Ctrl+V
删除(D)
Remove from Context Ctrl+Alt+Shift+向下
移动(V)...
重命名(M)... F2
导入(I)...
导出(O)...
刷新(F) F5
关闭项目(S)
关闭不相关的项目(U)
外部工具(X)
运行方式(R)
调试方式(D)
Team
比较对象(A)
替换为(L)
从本地历史记录复原(Y)...
属性(R) Alt+Enter
Commit... Ctrl+#
Fetch from Upstream
Push to Upstream
Remote
Switch To
Advanced
Pull
Synchronize Workspace
Merge Tool
Merge...
Reset...
Rebase...
Create Patch...
Apply Patch...
Ignore
Add to Index
Remove from Index
Untrack
Show in Repositories View
Show in History
Disconnect
Fetch From...
Push...
Fetch from Gerrit...
Push to Gerrit...
Configure Fetch from Upstream...
Configure Push to Upstream...
新建Web项目
新建移动A
了解HBuilder | 代码助手，飞一样的编码 | 绿

- WebStrom 集成 GIT 开发
  - 自主配置完成
  - webstrom 集成 git 提示
    - webstrom 已经集成了 git 插件
    - 需要在 PC 上安装 git 客户端
    - webstrom 中需要配置 git.exe 可执行命令路径

1-2.17 项目案例

- JQuery 项目提交 GIT 共享
- JQuery 项目修改提交
- 阶段项目开发