

git 使用简用指南

田宇伟 (fishtai0)

Apr 28, 2017

“如果你严肃对待编程，就必定会使用‘版本管理系统’
(Version Control System, VCS) 。”

为什么是git？

- 商业与开源版本管理的事实标准
- 所有平台对git提供图形用户界面支持、IDE的支持
- 令人难以置信的非线性分支管理系统
- 完全分布式
- 设计简单
- 速度飞快

安装git



Linux



Mac OS X



Windows

<https://git-scm.com/downloads>

<https://github.com/git/git/blob/master/INSTALL>

安装git



SourceTree

<https://sourcetreeapp.com>

git首次运行前配置

配置变量存储位置：

1. `/etc/gitconfig`
2. `~/.gitconfig` 或 `~/.config/git/config`
3. 当前仓库 `.git/config`

每一个级别覆盖上一级别的配置

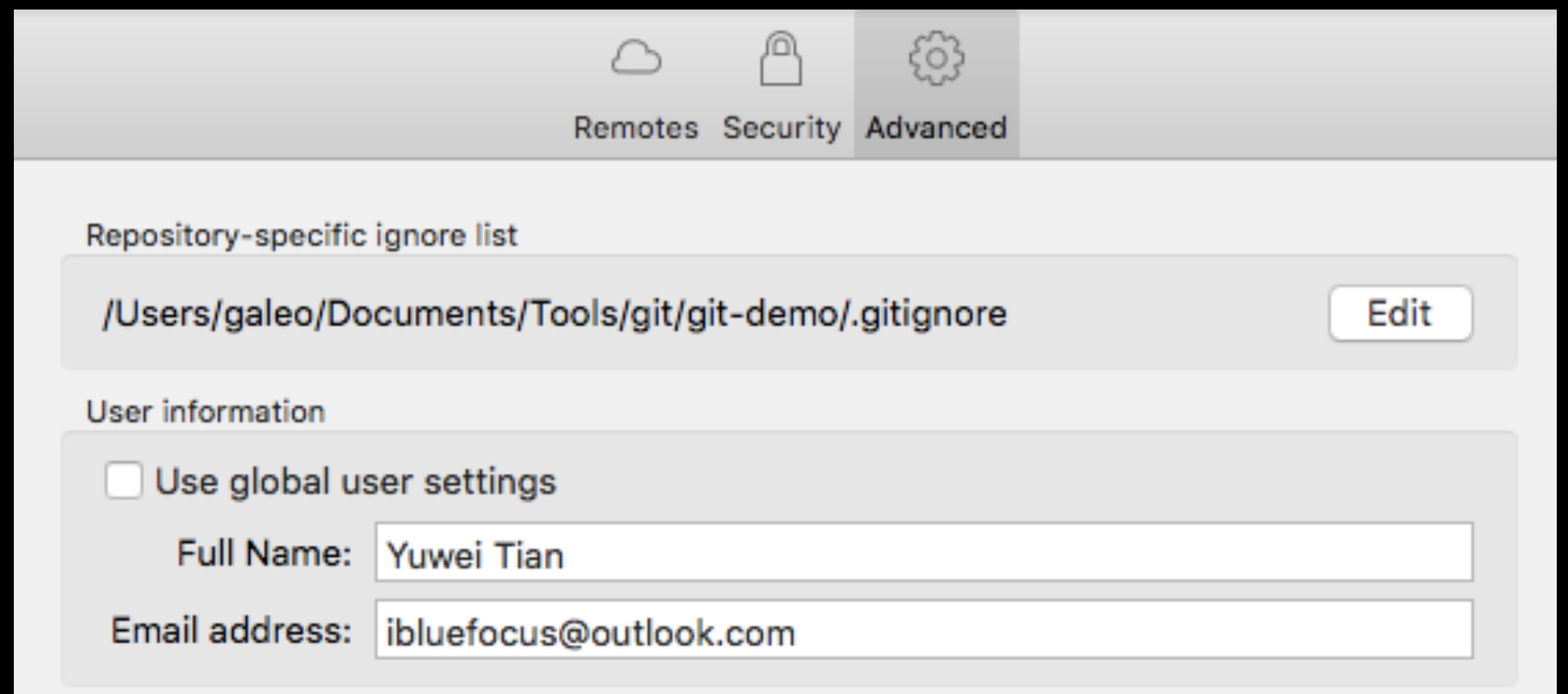
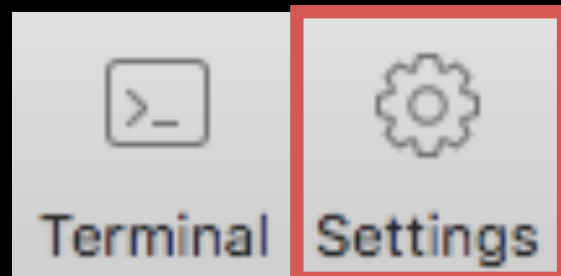
git配置文件采用INI文件格式

git首次运行前配置

用户信息（用户名和邮箱地址）

```
$ git config --global user.name "John Doe"  
$ git config --global user.email johndoe@example.com
```

如果想针对特定项目使用不同的用户名称与邮件地址，可以在那个项目目录下运行没有 `--global` 选项的命令来配置。



git首次运行前配置

文本编辑器

```
$ git config --global core.editor emacs
```

如果未配置，git 会使用操作系统默认的文本编辑器，通常是 Vim。

获取git帮助

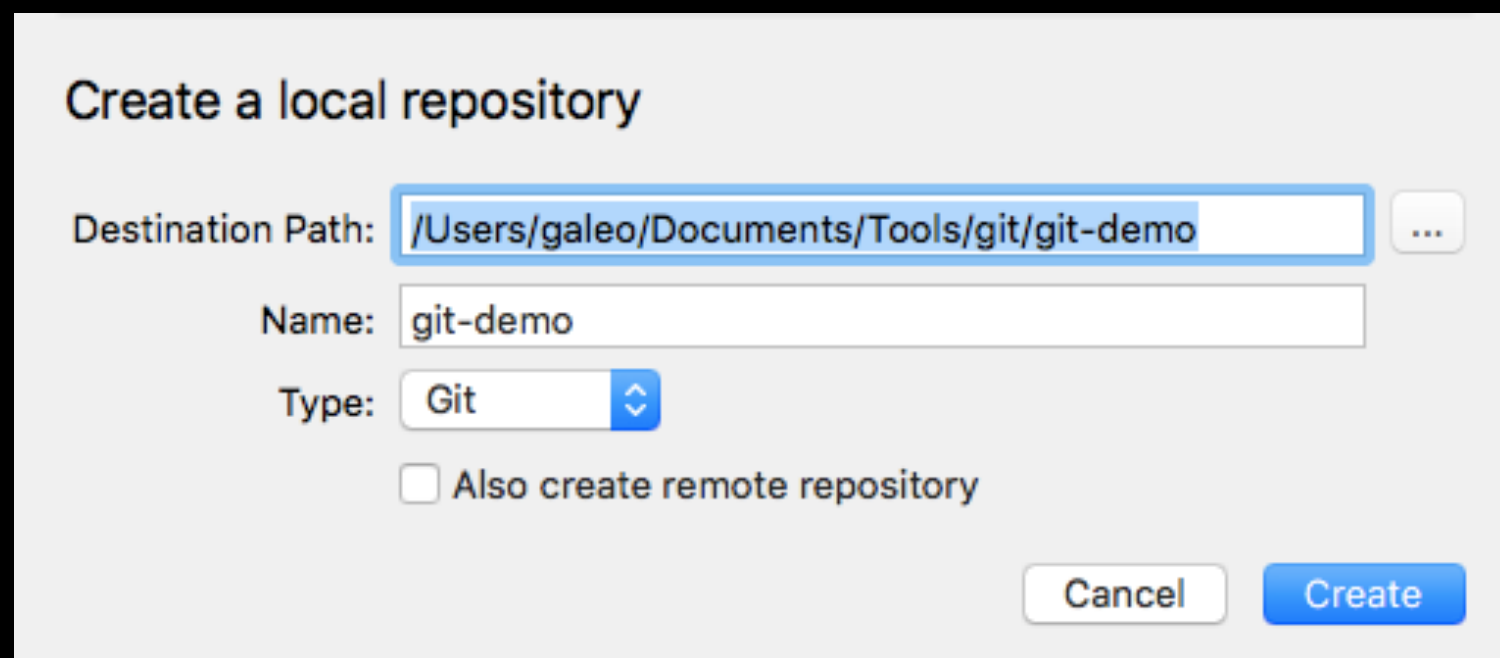
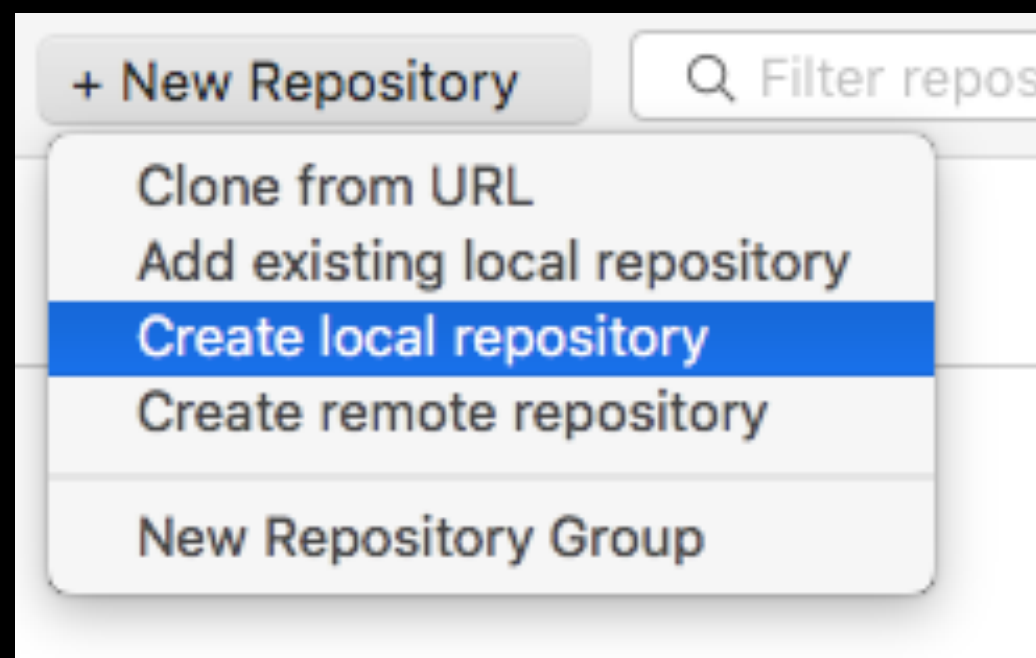
3种方法：

```
$ git help  
$ git --help  
$ man git
```

eg., 获得 `config` 命令的手册: `$ git help config`

创建新git仓库

创建新文件夹，打开，然后执行
`git init`
以创建新的git仓库。

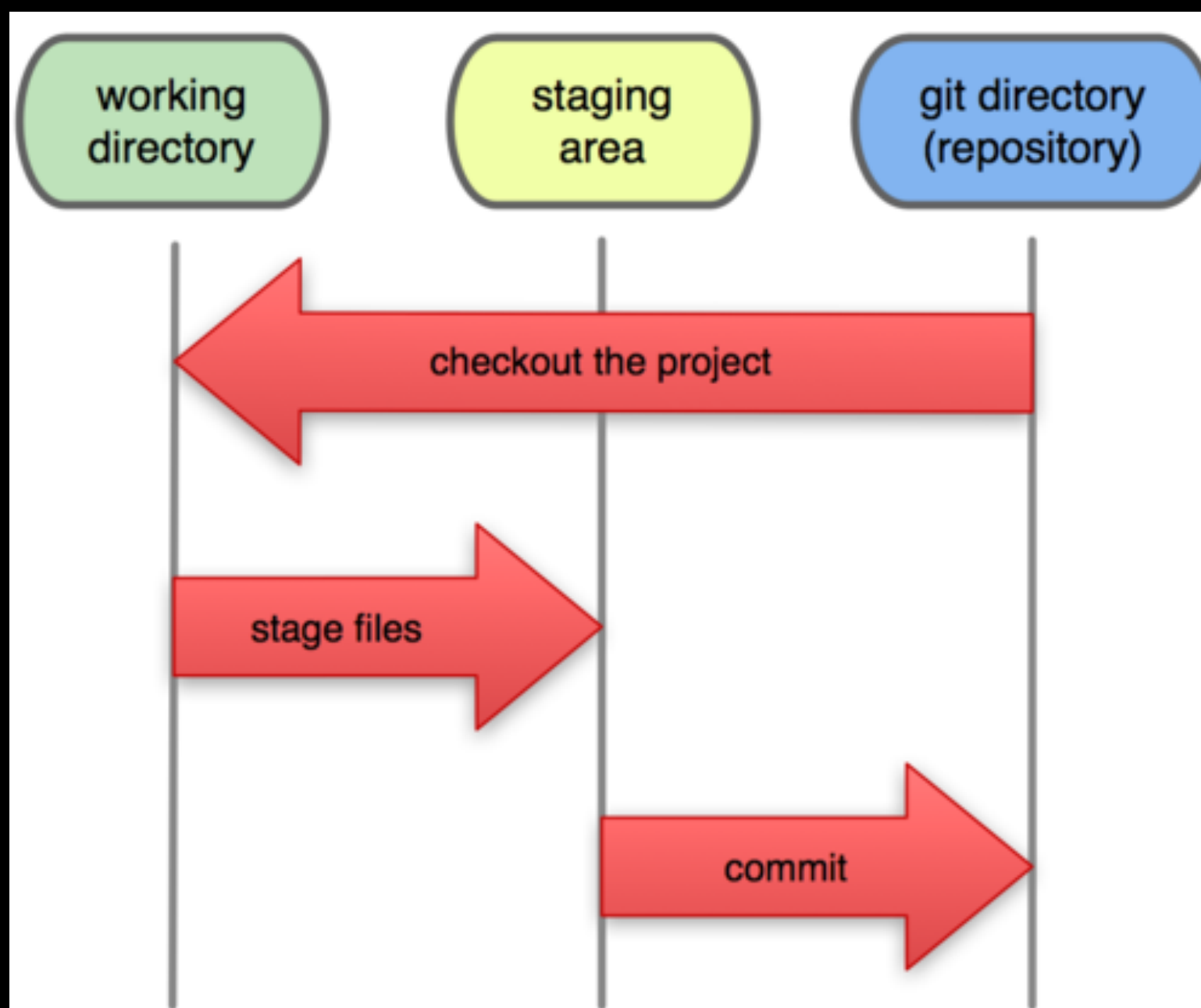


git的工作区、暂存区、版本库

工作区

暂存区

版本库



git添加和提交

提出更改（把它们添加到暂存区）：

```
git add <filename>
```

```
git add *
```

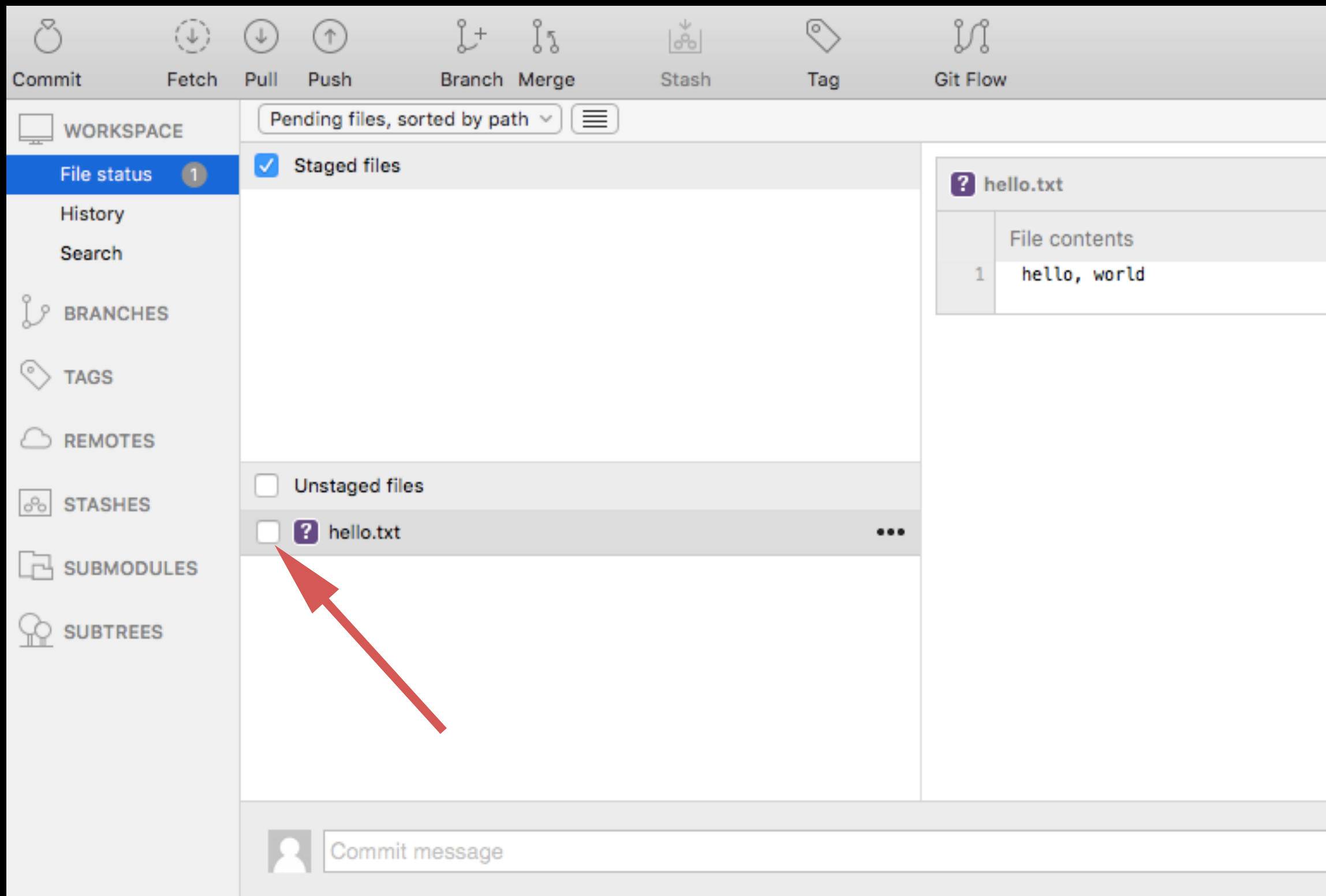
这是 git 基本工作流程的第一步；

实际提交改动：

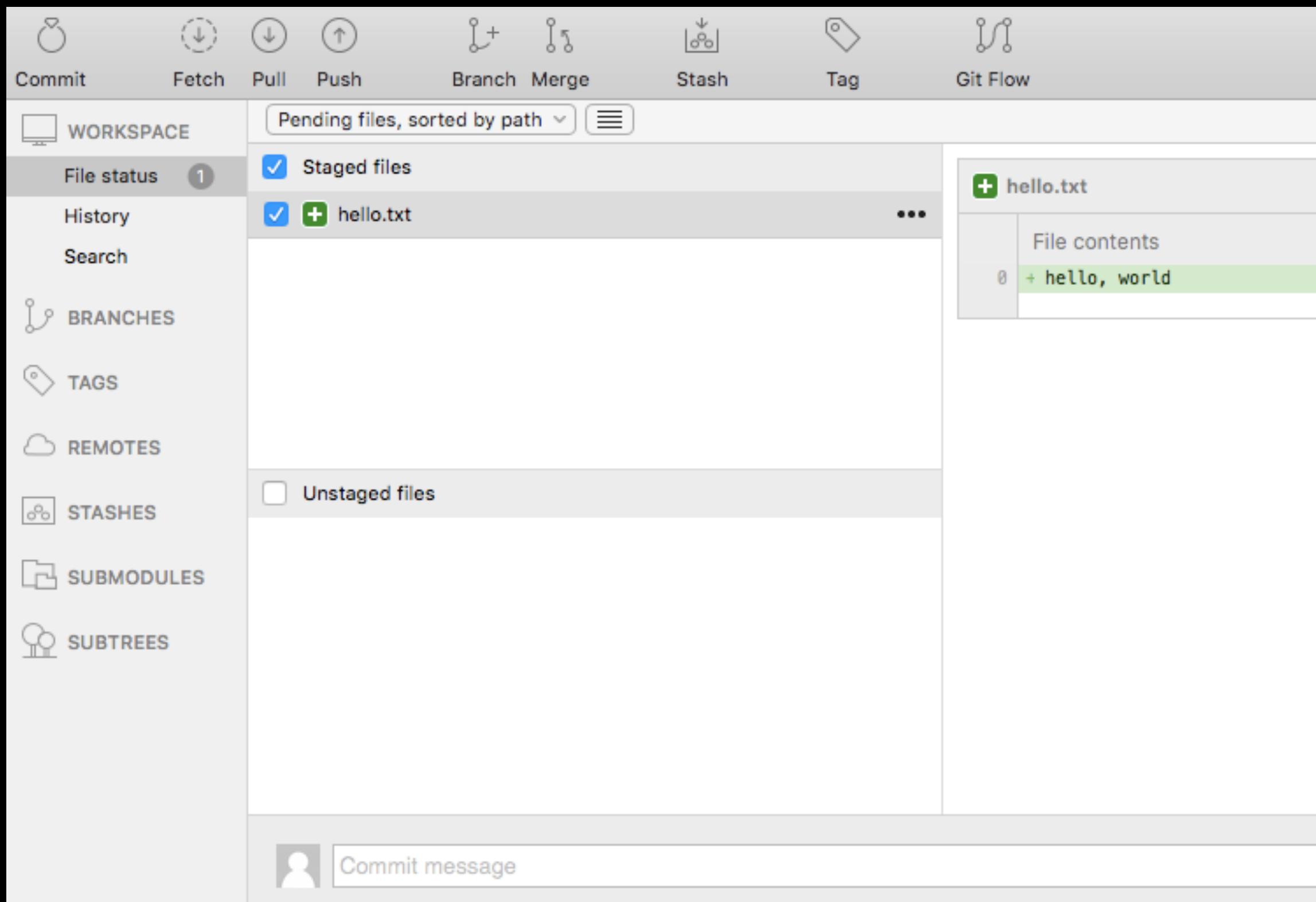
```
git commit -m "commit message"
```

```
$ echo "Hello, world" > hello.txt
$ git add hello.txt
$ git ci -m "init commit"
[master (root-commit) b497f22] init commit
1 file changed, 1 insertion(+)
create mode 100644 hello.txt
```

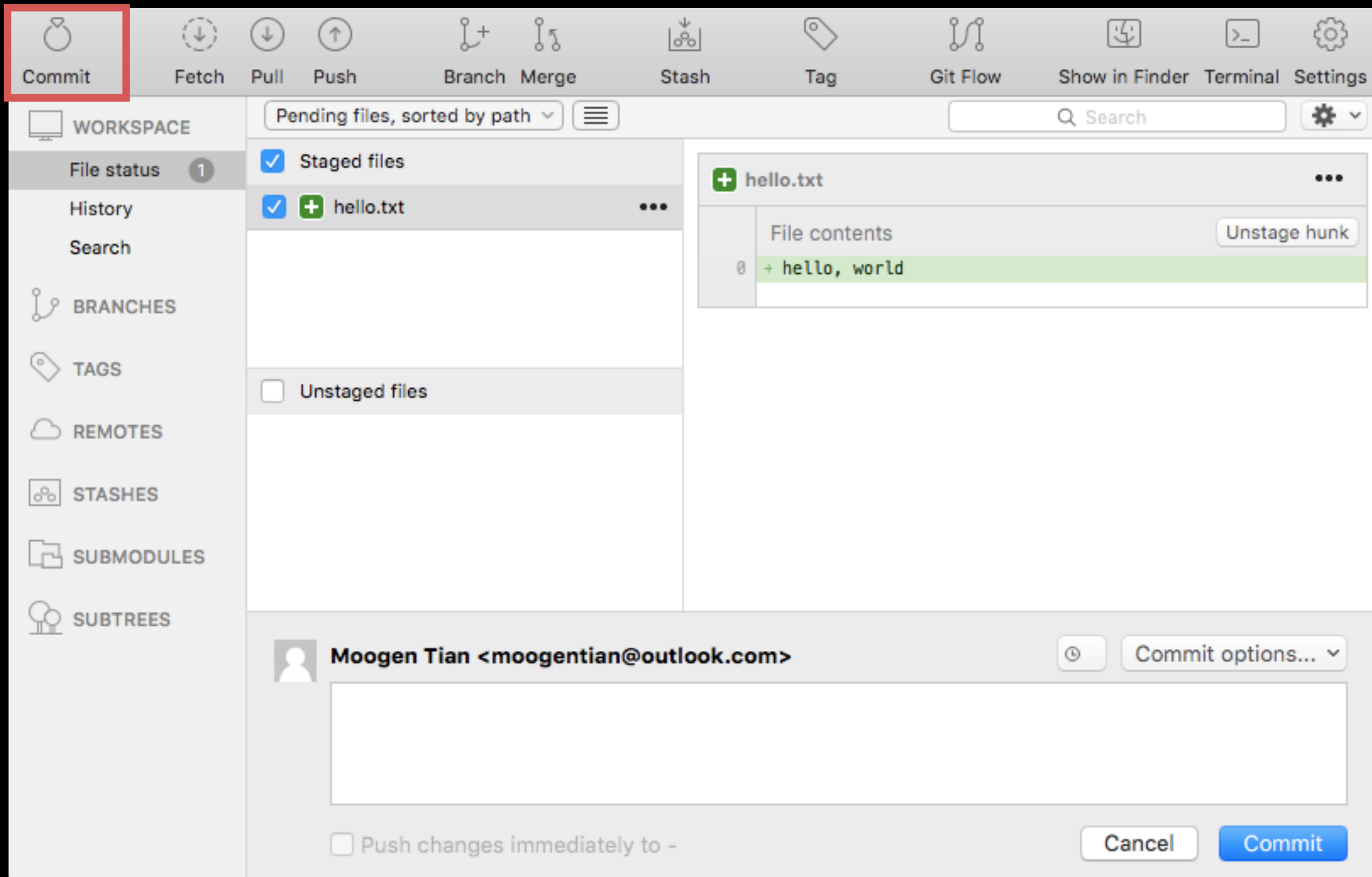
git添加和提交



git添加和提交



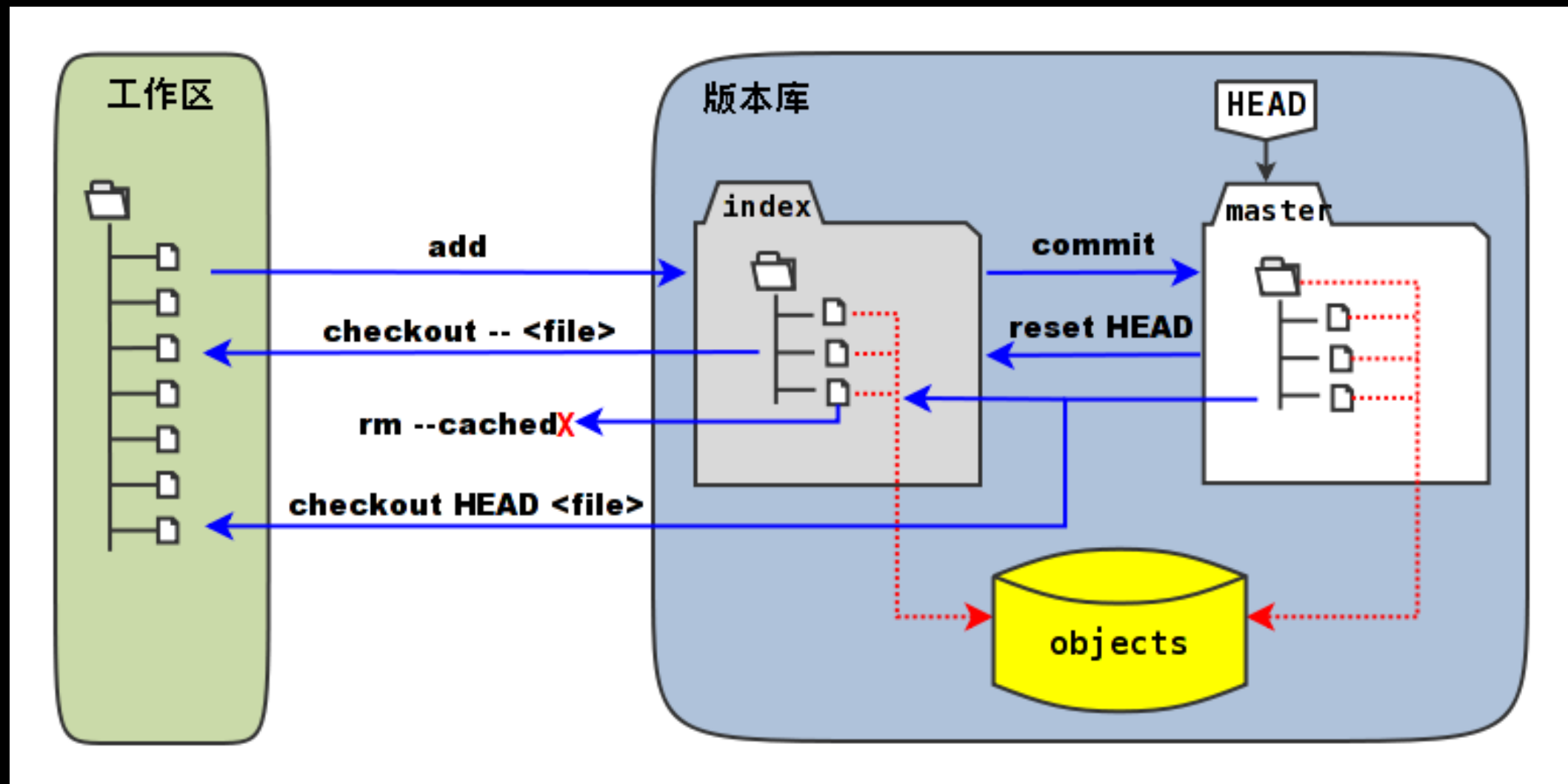
git添加和提交



git添加和提交

- 提交时记录的是放在暂存区的快照
- 任何还未暂存的仍然保持已修改状态，可以在下次提交时纳入版本管理
- 每一次运行提交操作，都是对你项目作一次快照，以后可以回到这个状态，或者进行比较

3种文件状态



- **已提交(committed):** 数据已经安全的保存在本地版本库中
- **已暂存(staged):** 对一个已修改文件的当前版本做了标记, 使之包含在下次提交的快照中
- **已修改(modified):** 修改了文件, 但还没保存到版本库中

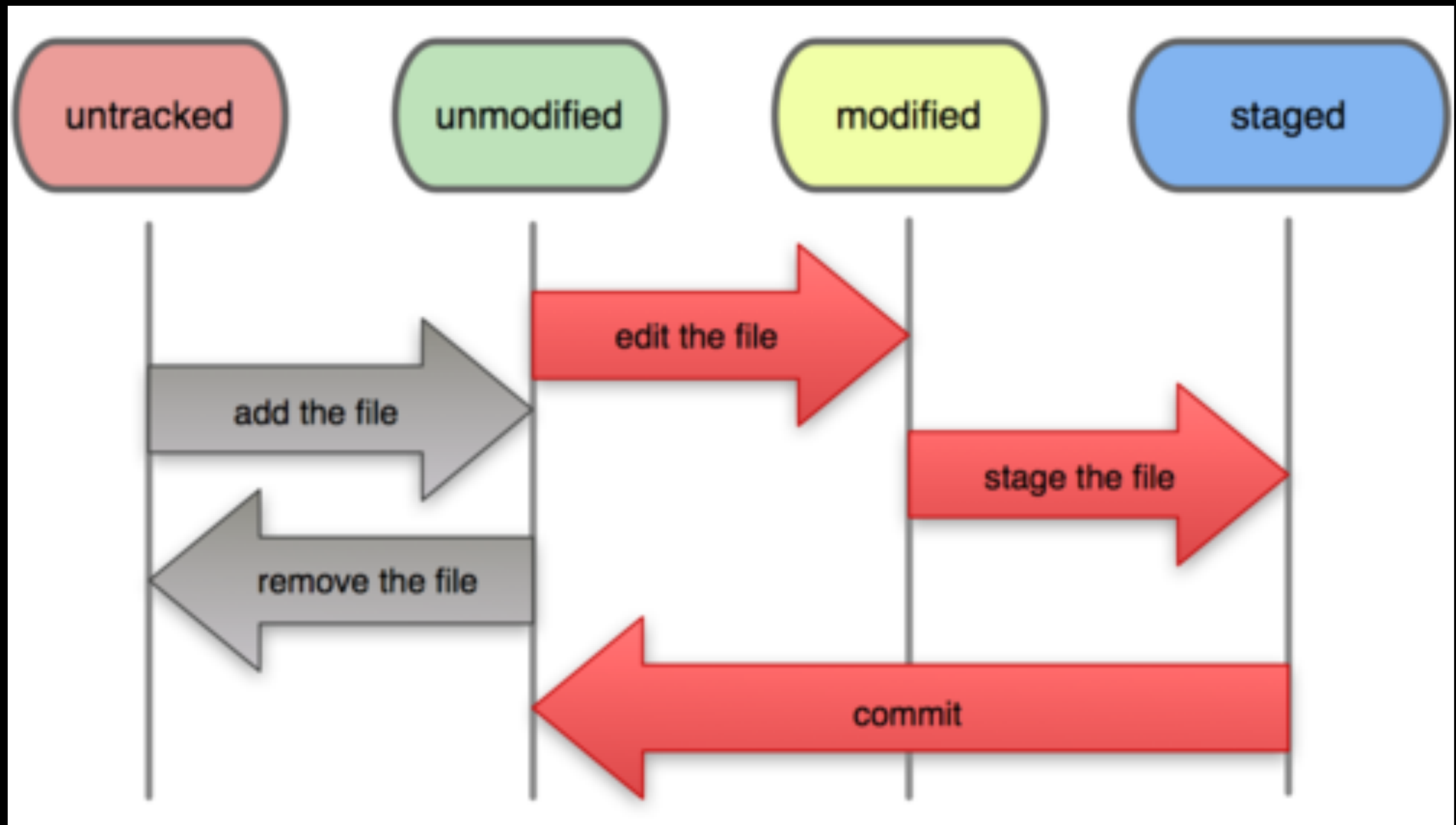
文件的生命周期

未跟踪

未修改

已修改

已暂存



使用 `git status` 来检查文件状态

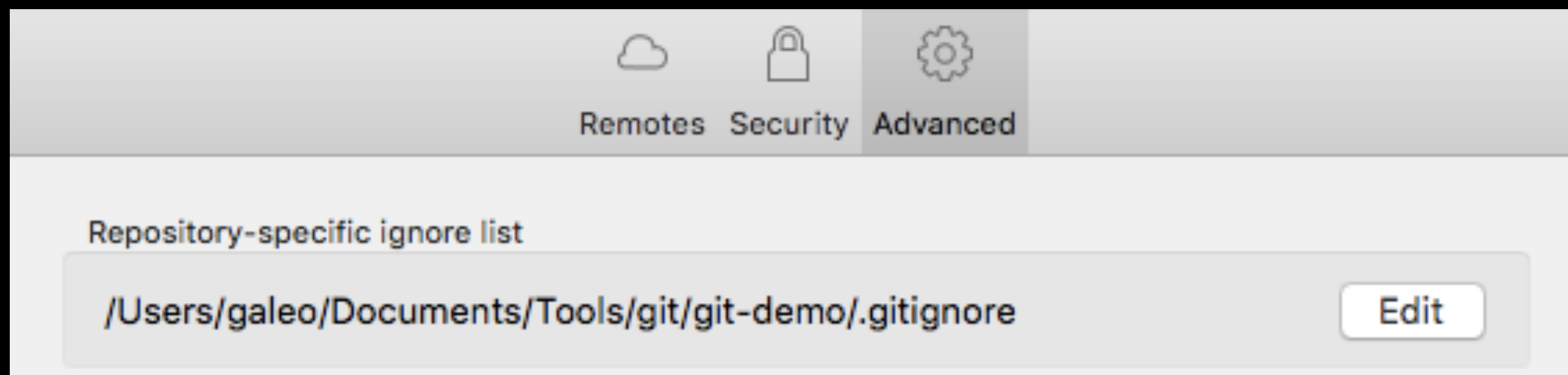
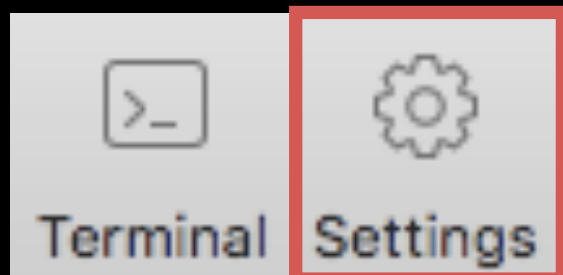
忽略文件

使用 `.gitignore` 文件来忽略不想跟踪的文件

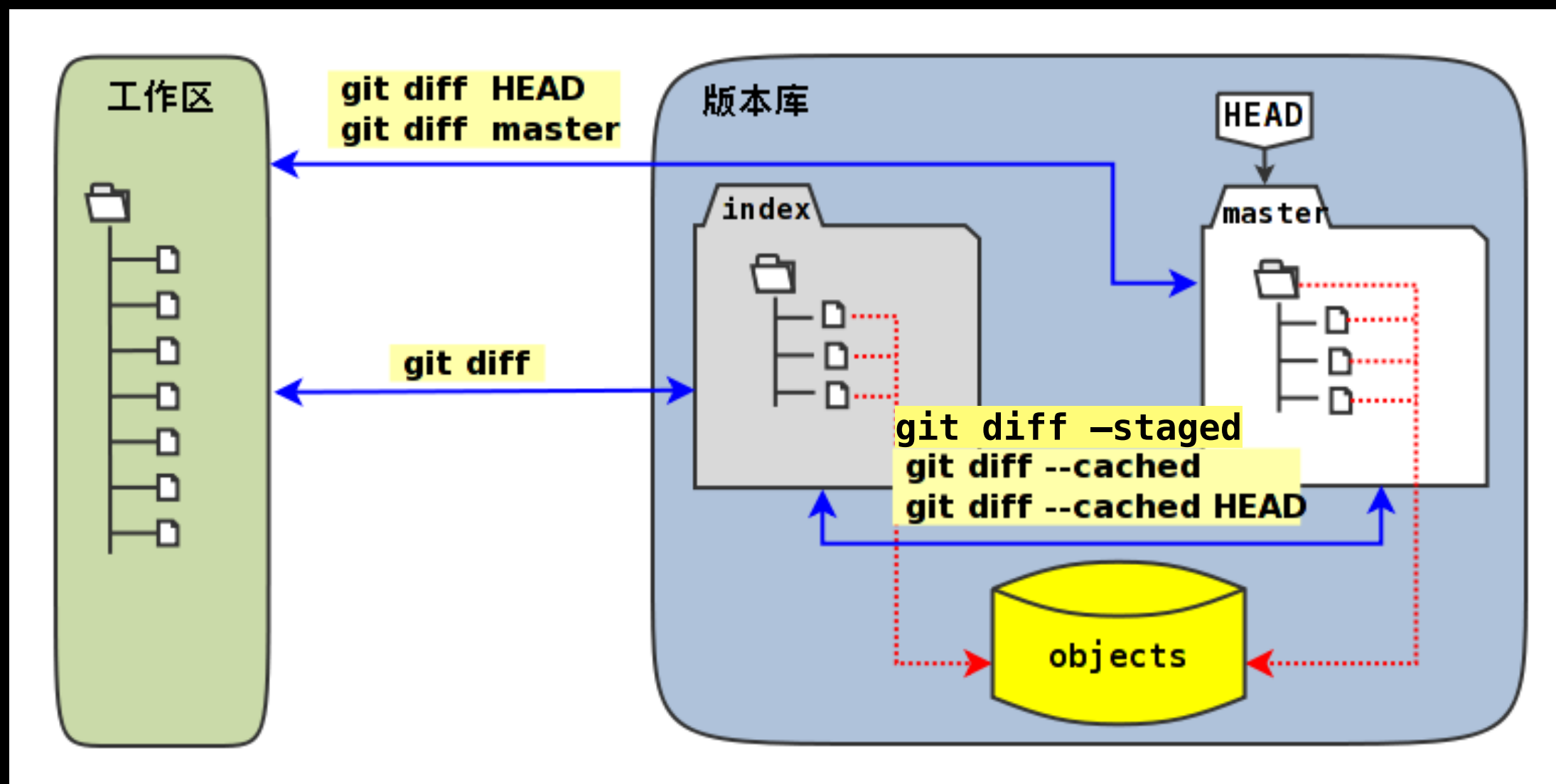
文件 `.gitignore` 的格式规范

eg.: <https://github.com/github/gitignore>

忽略文件



查看变动



使用不同的参数调用`git diff`命令，可以对工作区、暂存区、HEAD中的内容两两比较。

查看变动

WORKSPACE

Pending files, sorted by path

File status 1

History

Search

BRANCHES

TAGS

REMOTES

STASHES

☒ Staged files

☒ + hello.txt

☐ Unstaged files

☐ ... hello.txt

hello.txt

Hunk 1 : Lines 1-2

Stage hunk Discard hunk

1 1 hello, world

2 + Hello, China

WORKSPACE

Pending files, sorted by path

File status 1

History

Search

BRANCHES

TAGS

REMOTES

☒ Staged files

☒ + hello.txt

☐ Unstaged files

☐ ... hello.txt

+ hello.txt

File contents

Unstage hunk

0 + hello, world

git分支

把你的工作从开发主线上分离开来，以免影响开发主线。

git鼓励在工作流程中频繁地使用分支与合并。

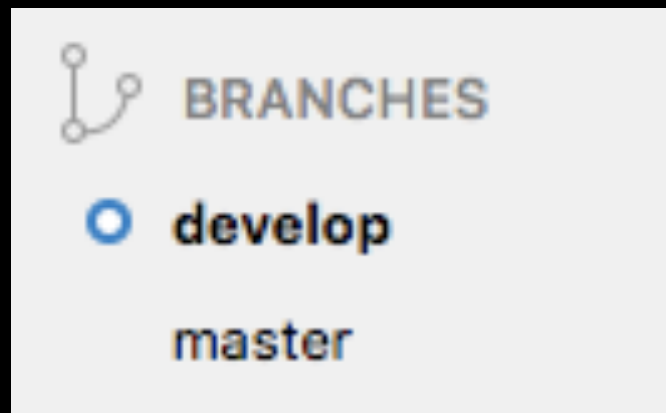
git的分支，本质上仅仅是指向**提交对象的可变指针**。

默认分支名字是 `master`

git分支

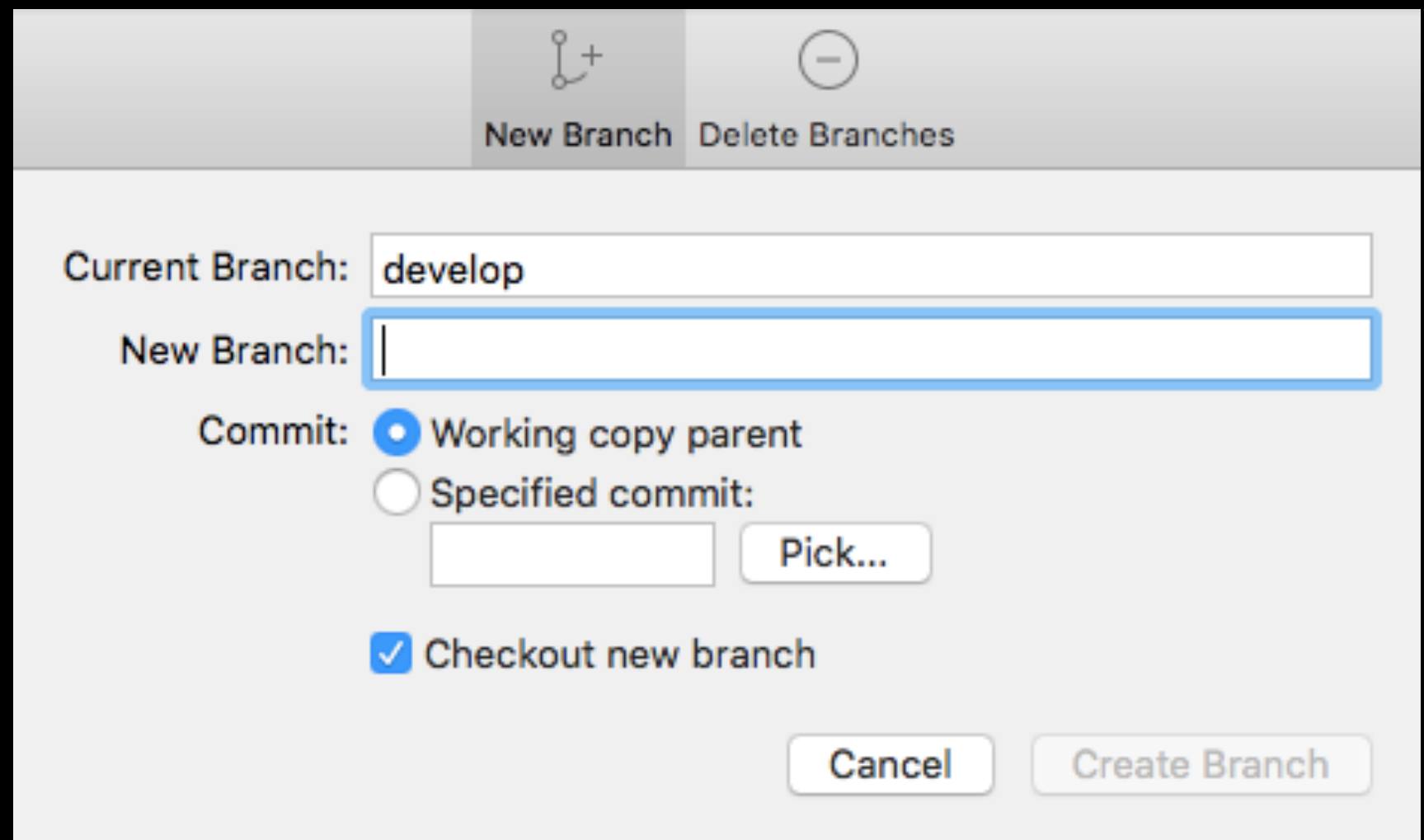
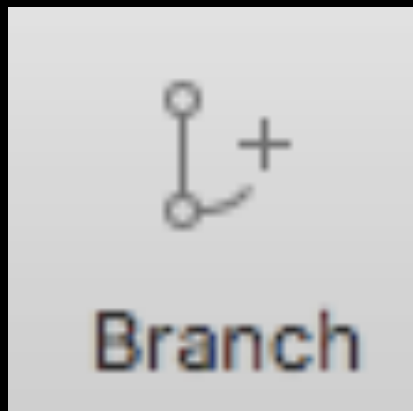
显示当前的工作分支 `git branch`

```
$ git branch  
develop  
feature-a  
* master
```



git分支

分支创建 `git branch <new-branch-name>`

A screenshot of a "New Branch" dialog box. The dialog has a title bar with two tabs: "New Branch" (active, with a branch icon) and "Delete Branches" (with a minus icon). The main area contains the following fields and options:

- "Current Branch:" with a text field containing "develop".
- "New Branch:" with an empty text field.
- "Commit:" section with two radio buttons:
 - ☒ "Working copy parent"
 - ☐ "Specified commit:" followed by an empty text field and a "Pick..." button.
- A checked checkbox labeled "Checkout new branch".
- At the bottom right, two buttons: "Cancel" and "Create Branch".

git分支切换

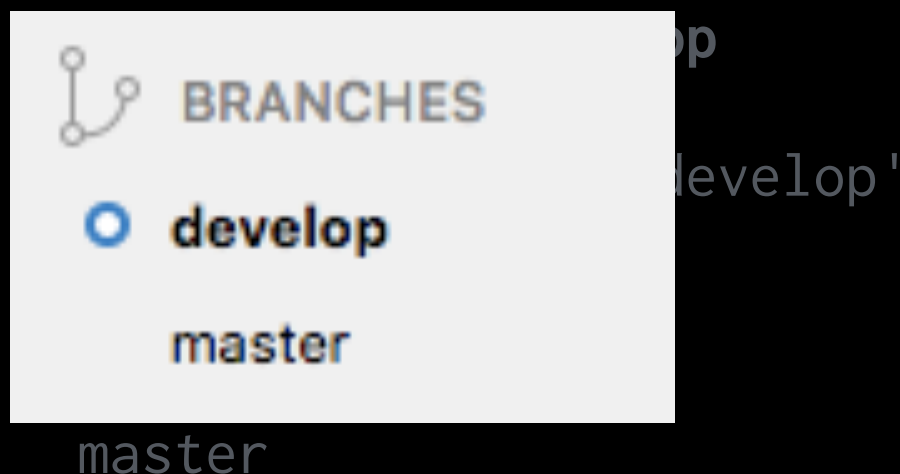
```
git checkout <branch-name>
```

HEAD 分支随着提交操作自动向前移动。

检出时 HEAD 随之移动。

分支切换会改变你工作目录中的文件。如果 git 不能干净利落地完成这个任务，它将禁止切换分支。

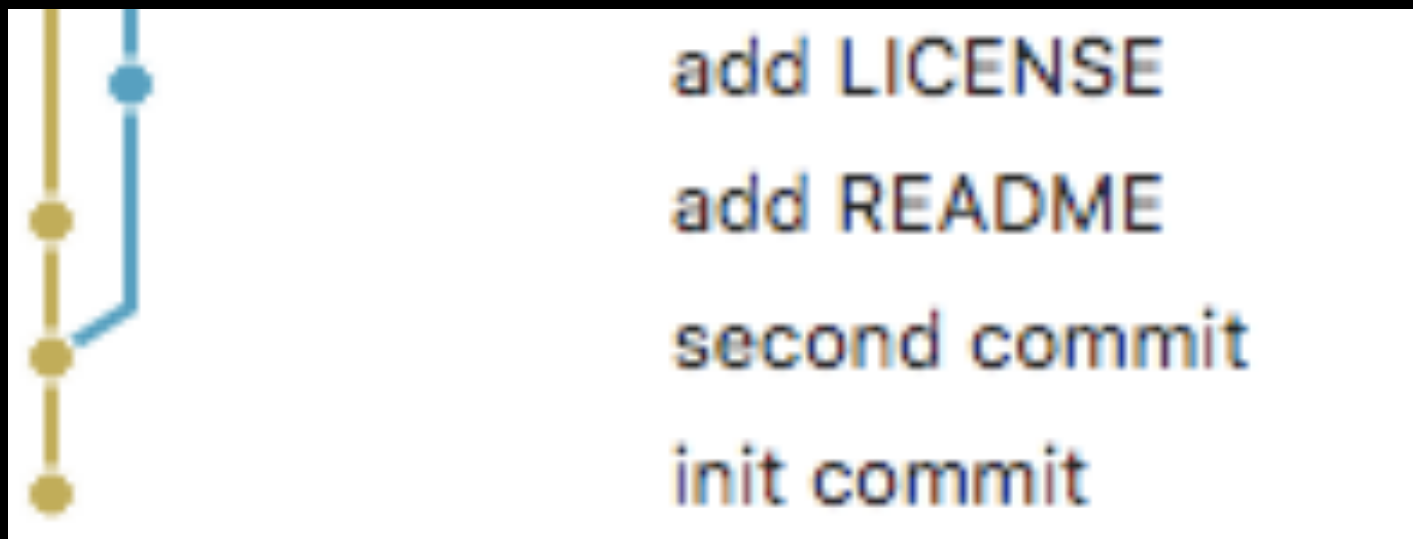
你可以在不同分支间不断地来回切换和工作，并在时机成熟时将它们合并起来。



git分支分叉历史

```
git log --oneline --decorate --graph --all
```

```
$ git log --oneline --decorate --graph --all
* 3570d25 (HEAD -> develop) add LICENSE
| * c876ccf (master) add README
|/
* be6f970 second commit
* b497f22 init commit
```

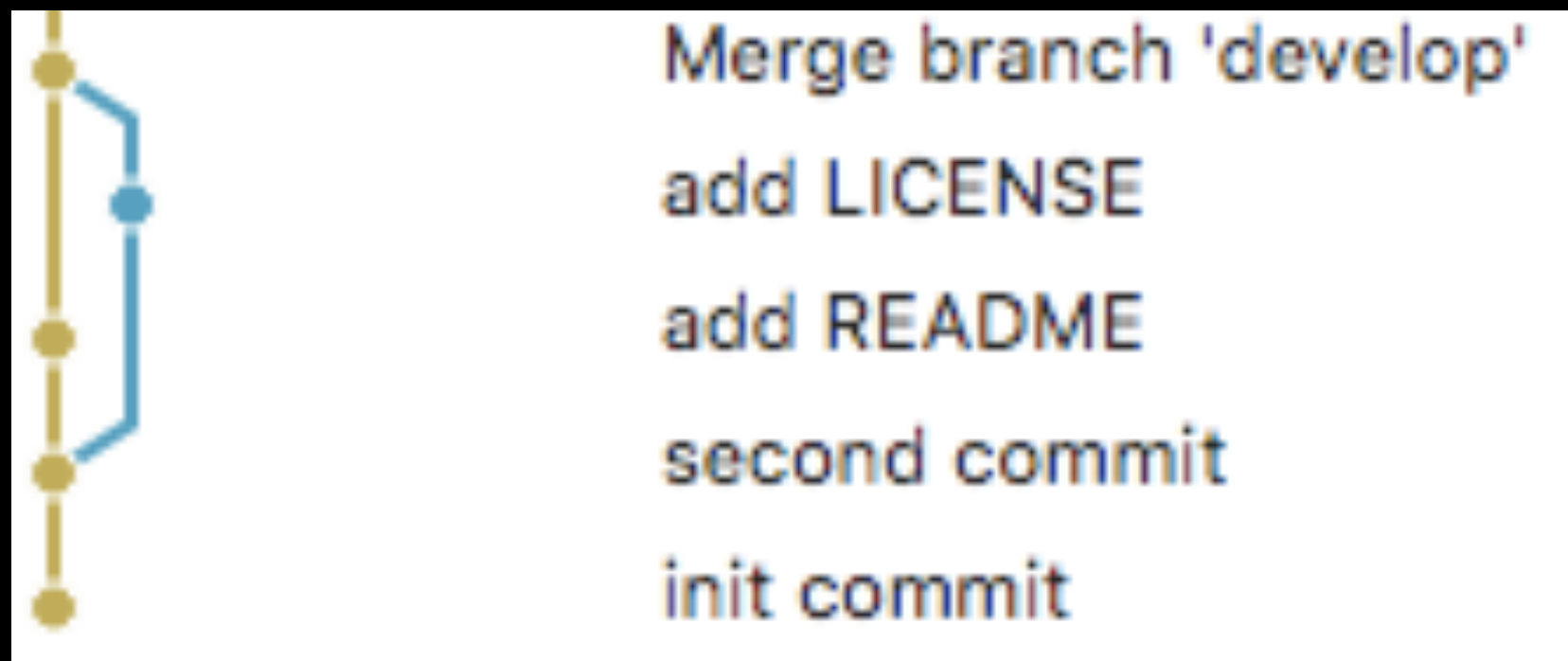


分支合并

检出到你想合并入的分支，然后运行 `git merge` 命令。

```
$ git checkout master
Switched to branch 'master'
$ git merge develop
Merge made by the 'recursive' strategy.
 LICENSE | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 LICENSE
```

分支合并



git 会使用两个分支的末端所指的快照，以及这两个分支的共同祖先，做一个简单的三方合并。

合并提交 不止一个父提交

遇到冲突时的分支合并

在两个不同的分支中，对同一个文件的同一个部分进行了不同的修改，git 就没法干净的合并它们。

先解决合并冲突后再合并。

git分支管理

查看每一个分支的最后一次提交 `git branch -v`

查看哪些分支已经合并到当前分支 `git branch --merged`

查看所有包含未合并工作的分支 `git branch --no-merged`

分支删除 `git branch -d <branch-name>`

想要删除分支并丢掉还未合并的工作，使用 `-D` 选项强制删除

git远程操作

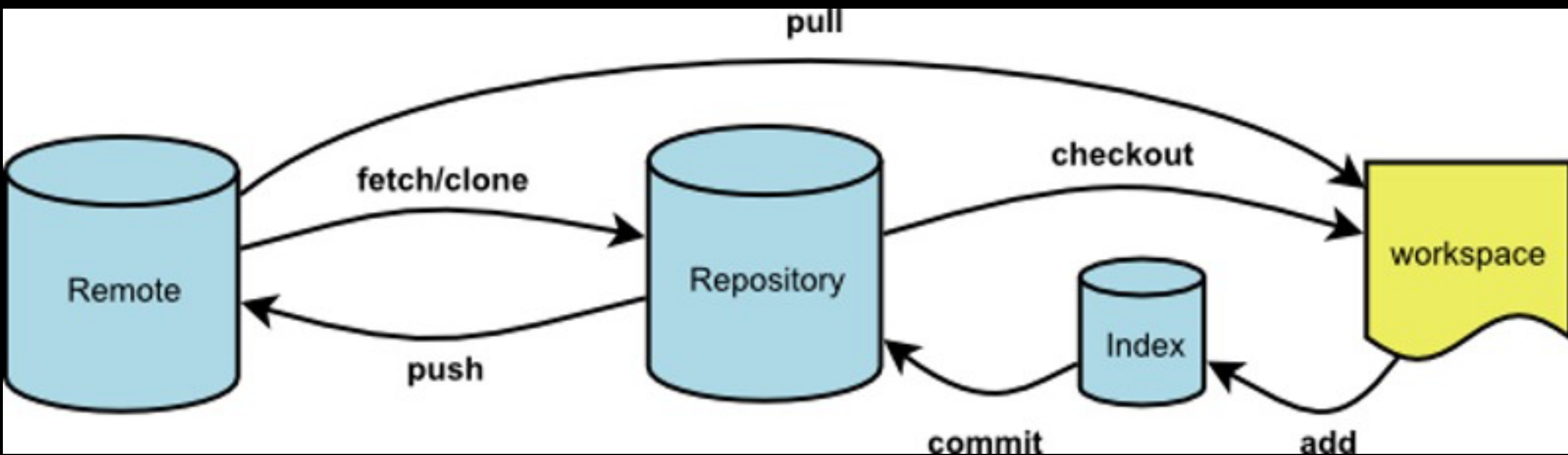
git clone

git remote

git fetch

git pull

git push



克隆一个git仓库

执行如下命令以创建一个本地仓库的克隆版本：

```
git clone /path/to/repository
```

如果是远端服务器上的仓库，你的命令会是这个样子：

```
git clone <版本库网址> [<本地目录名>]
```

Clone a repository

Cloning is even easier if you set up a [remote account](#).

Source URL:

Destination Path:



Name:

► Advanced Options

⚠ This is not a valid source path / URL

Cancel

Clone

远程仓库

git remote 管理主机名

每个远程主机都必须指定一个主机名。

列出所有远程主机: `git remote`

参看远程主机网址: `git remote -v` 默认主机名 `origin`

`git clone -o <name> <repository>`

查看某一主机的详细信息: `git remote show [remote-name]`

添加远程主机: `git remote add <shortname> <url>`

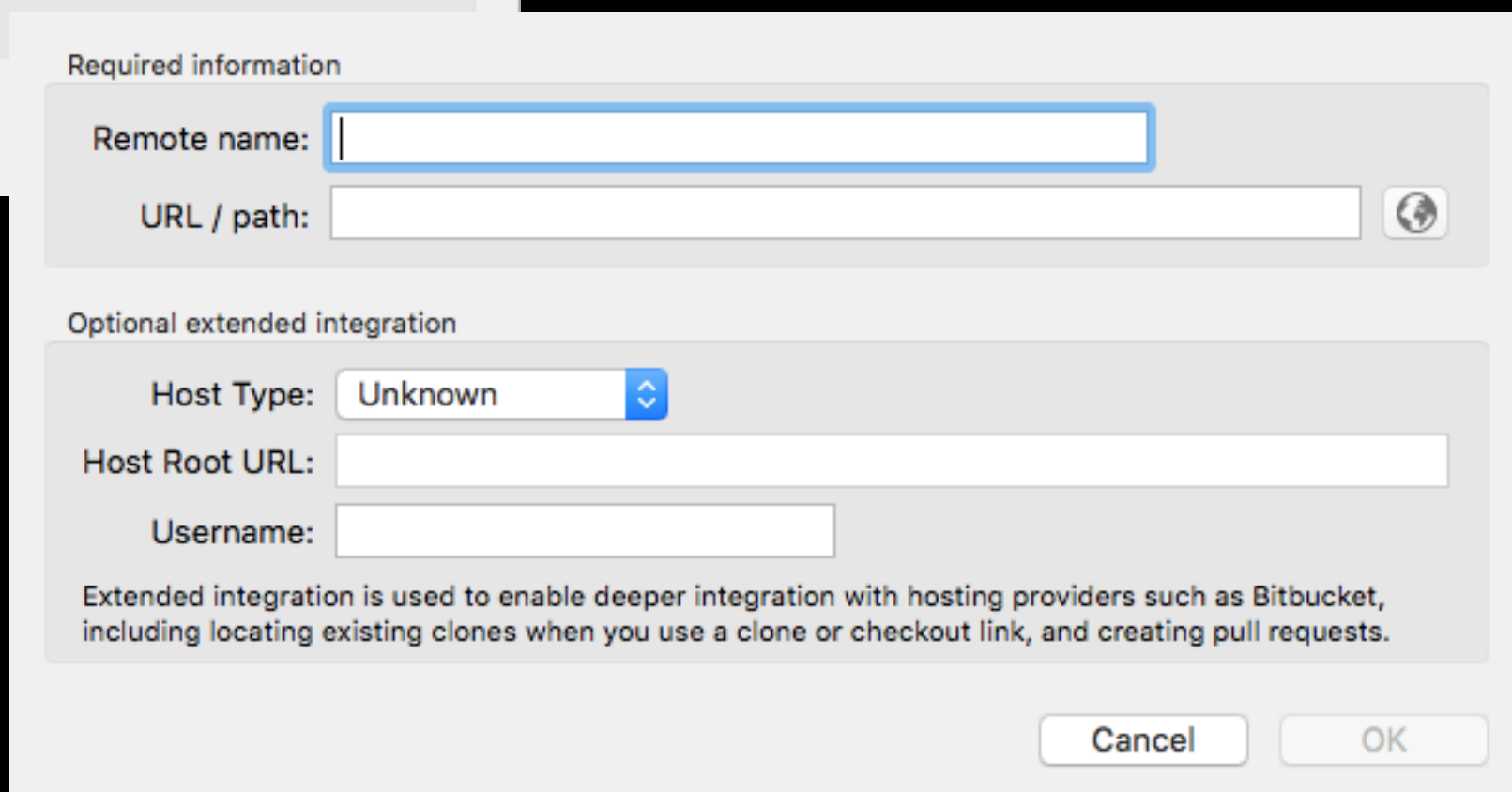
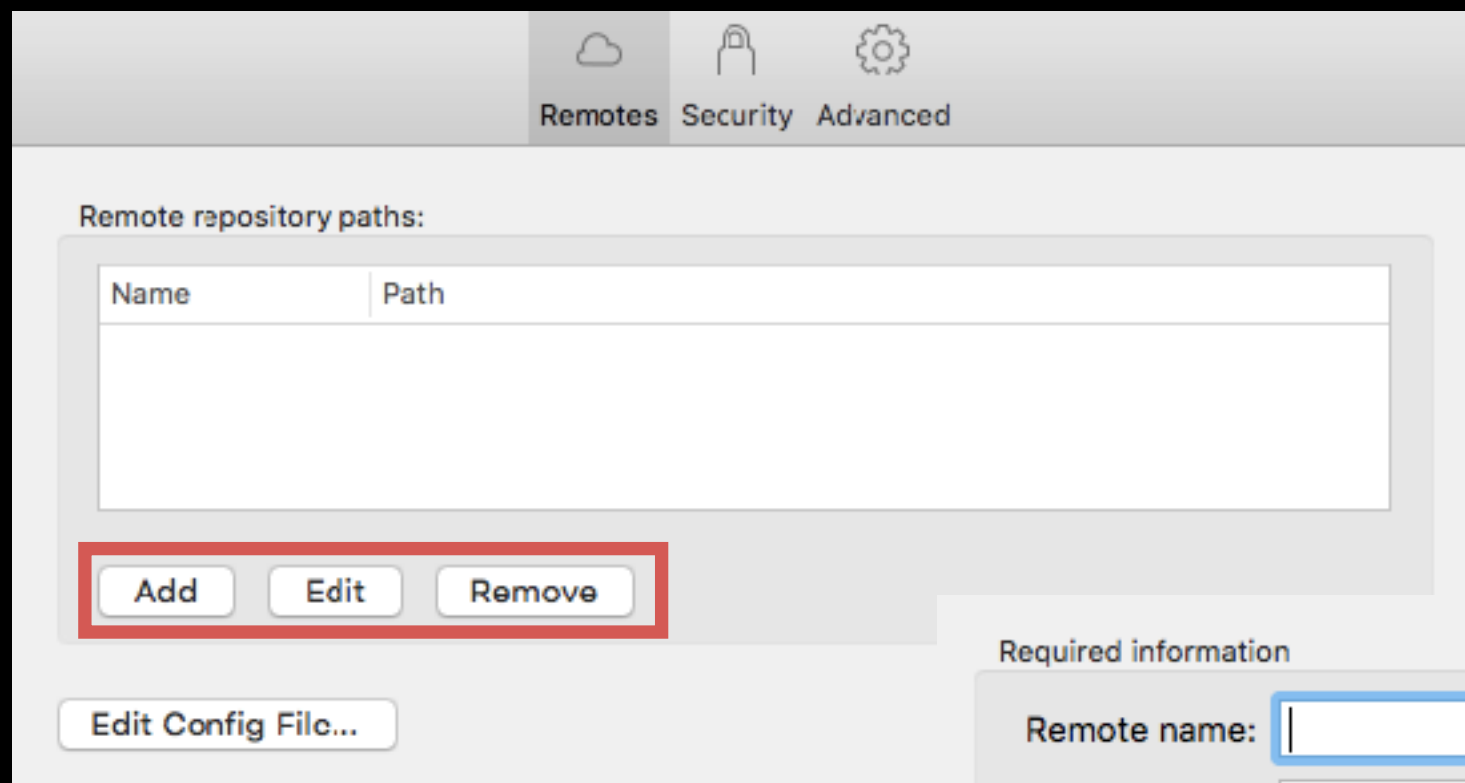
使用字符串简写来代替整个URL

删除远程主机: `git remote rm <remote_short_name>`

重命名远程主机: `git remote rename <old_name> <new_name>`

远程仓库

git remote 管理主机名



从远程仓库中抓取

`git fetch`

将某个远程主机的更新，全部取回本地：

```
git fetch [remote-name]
```

通常用来查看其他人的进程，取回的代码
对本地的开发代码没有影响。

取回特定分支的更新：

```
git fetch <remote-name> <branch-name>
```

取回的更新，在本地主机上要用“**远程主机名/分支名**”的形式读取：

`origin/master`

查看远程分支：`git branch -r`

取回更新后，在其基础上创建新分支：

```
git checkout -b newBranch origin/master
```

在本地分支上合并远程分支：`git merge origin/master`

从远程仓库中抓取

`git fetch`



Fetch

- ☒ Fetch from all remotes
- ☐ Prune tracking branches no longer present on remote(s)
- ☐ Fetch and store all tags locally

Cancel

OK

从远程仓库中抓取并合并本地分支

`git pull` 等同于先 `git fetch` 再 `git merge`

完整命令：

```
git pull <remote-name> <remote-branch-name>:<local-branch-name>
```

将远程分支与当前分支合并：

```
git pull <remote-name> <remote-branch-name>
```

默认情况下，`git clone` 命令会自动设置本地 `master` 分支跟踪克隆的远程仓库的 `master` 分支（或不管是什么名字的默认分支）。

从远程仓库中抓取并合并本地分支

`git pull` 等同于先 `git fetch` 再 `git merge`



Pull from repository: Custom

Remote branch to pull: Select a branch Refresh

Pull into local branch: master

Options

- ☒ Commit merged changes immediately
- ☐ Include messages from commits being merged in merge commit
- ☐ Create new commit even if fast-forward merge
- ☐ Rebase instead of merge (WARNING: make sure you haven't pushed your changes)

Cancel OK

将本地分支更新推送到远程主机

```
git push
```

完整命令：

```
git push <remote-name> <local-branch-name>:<remote-branch-name>
```

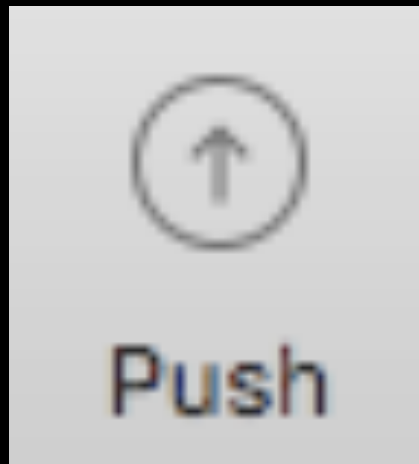
分支推送顺序：<来源地>:<目的地>


只有当你有所克隆服务器的写入权限，并且之前没有人推送过时，推送才能生效。

先将其他人的更新拉取下来并将其合并进你的工作后才能推送。

将本地分支更新推送到远程主机

git push



Push to repository: Custom 

Branches to push

Push?	Local branch	Remote branch	Track?
<input type="checkbox"/>			
<input type="checkbox"/>			
<input type="checkbox"/>			
<input type="checkbox"/>			
<input type="checkbox"/>			

☐ Select All

☒ Push all tags

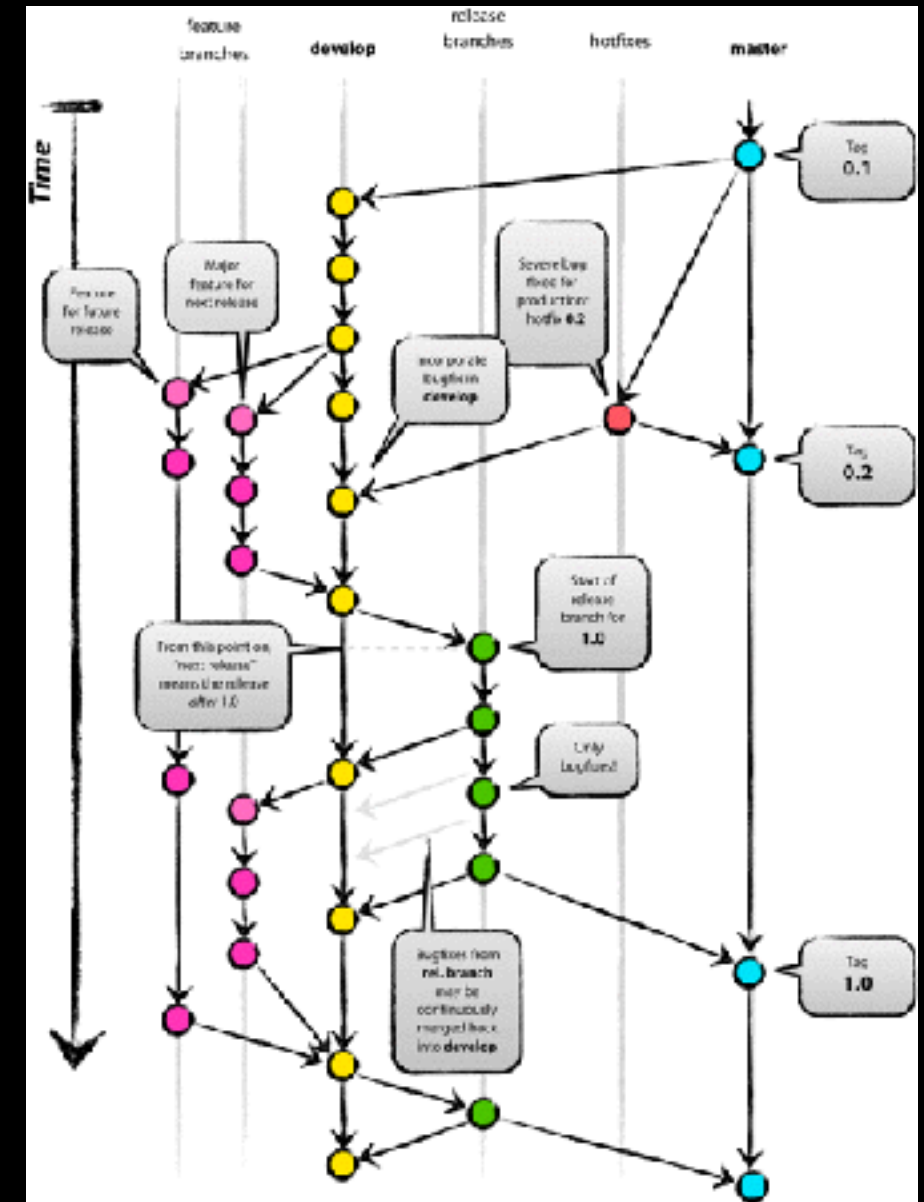
Cancel OK

git flow

Vincent Driessen提出的分支管理策略

A successful Git branching model, Jan 05, 2010

使版本库的演进保持简洁，主干清晰，
各个分支各司其职、井井有条



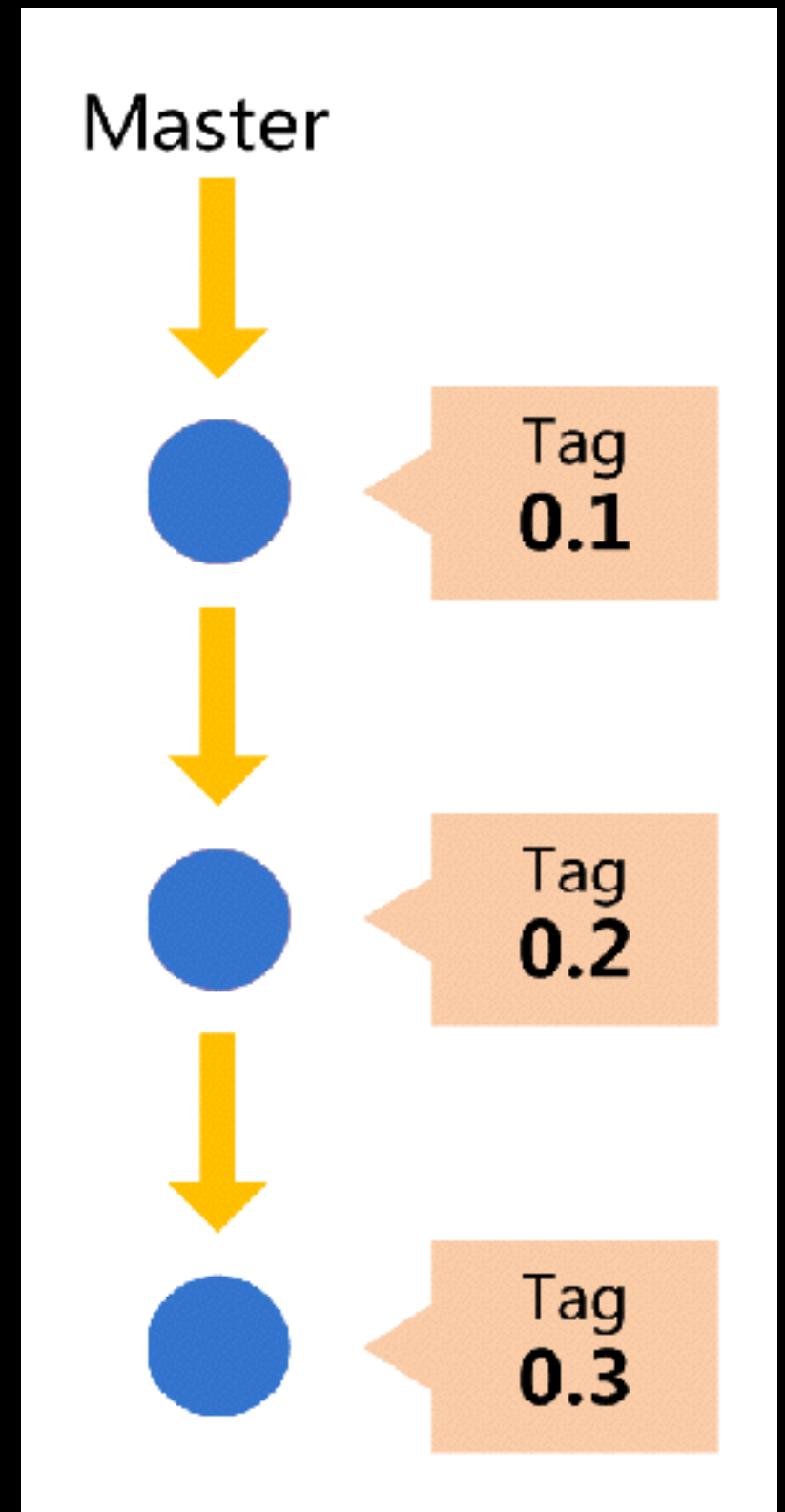
git flow流程图



git flow

主分支master

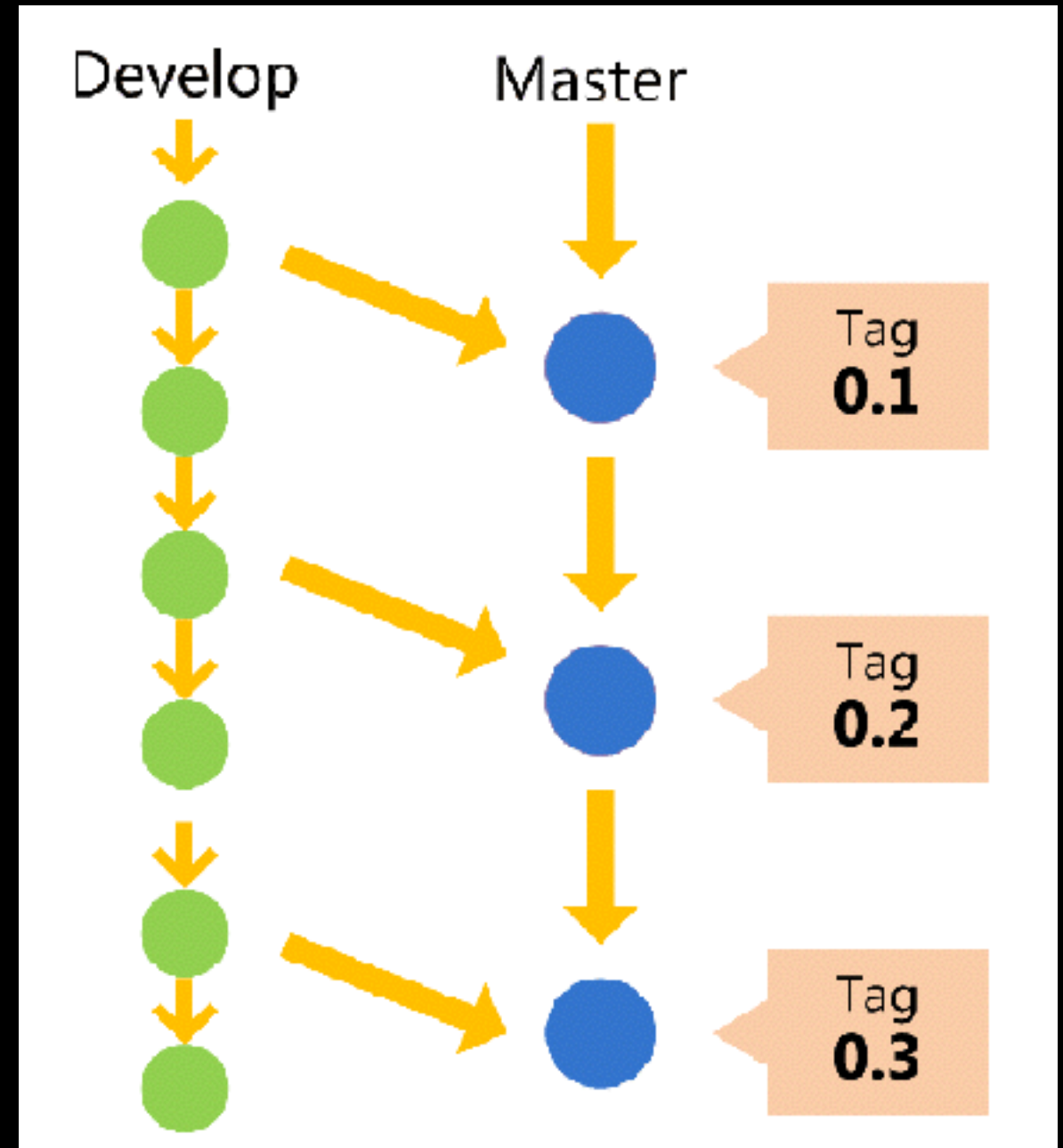
- 代码库应该有一个、且仅有一个主分支
- 所有提供给用户使用的正式版本，都在这个主分支上发布
- git主分支的名字，默认叫做master
- 主分支只用来发布重大版本



git flow

开发分支develop

- 开发用的分支，叫做 develop，日常开发应该在开发分支上完成
- 开发分支用来生成代码的最新隔夜版本（nightly）
- 正式对外发布，就在 master 分支上，对 develop 分支进行“合并”（merge）



git flow

开发分支develop

git创建develop分支的命令：

```
git checkout -b develop master
```

将develop分支发布到master分支的命令：

```
# 切换到Master分支  
git checkout master
```

```
# 对Develop分支进行合并  
git merge --no-ff develop
```

git flow

临时性分支

- 功能 (feature) 分支
- 预发布 (release) 分支
- 修补bug (hotfix) 分支

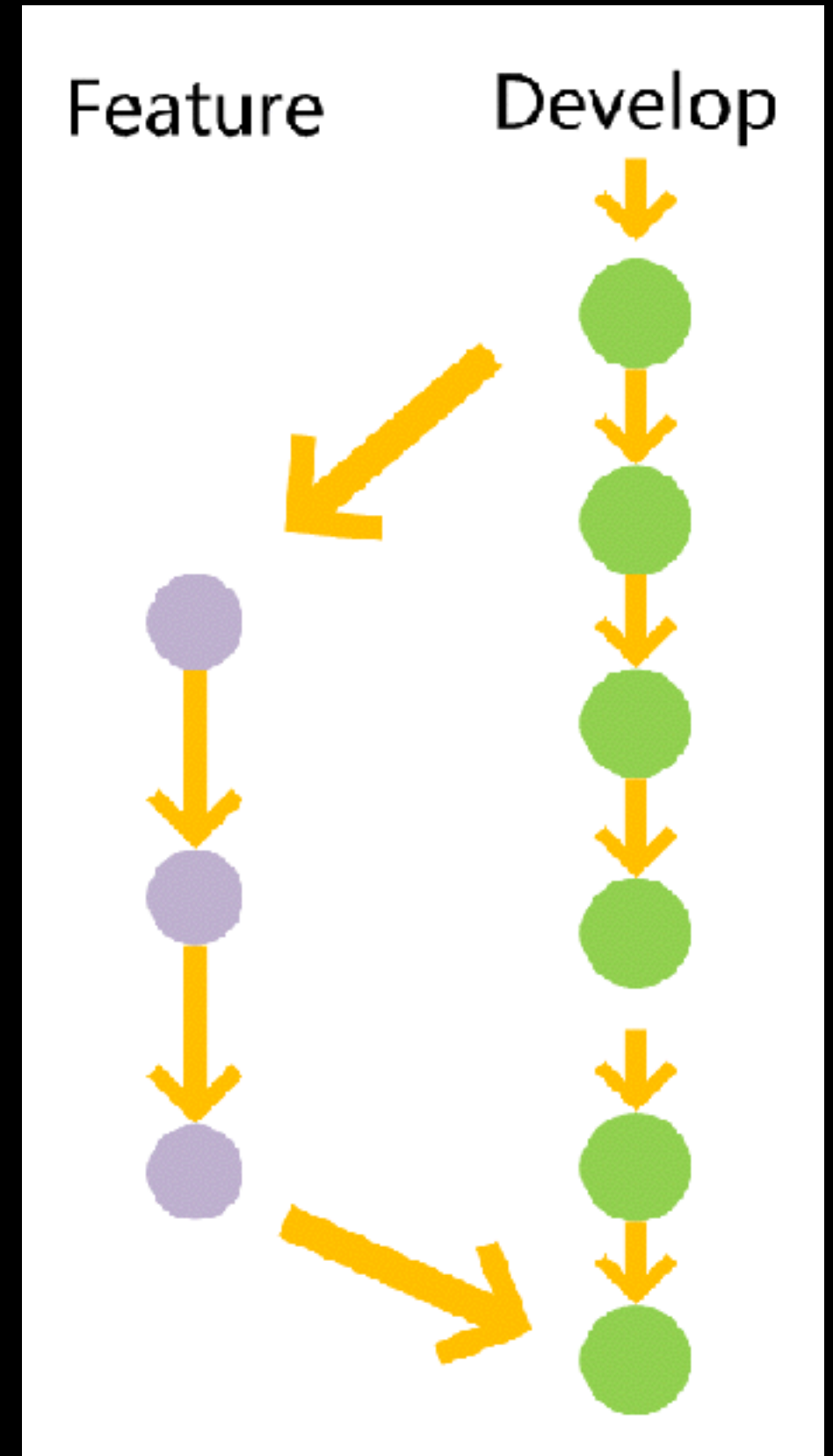
这三种分支都属于临时性需要，使用完以后，应该删除，使得代码库的常设分支始终只有master和develop。

git flow

功能（feature）分支

- 为了开发某种特定功能
- 从develop分支上分出来
- 开发完成后，要再并入develop

功能分支的名字，可以采用feature-*的形式命名。



git flow

功能 (feature) 分支

创建一个功能分支：

```
git checkout -b feature-x develop
```

开发完成后，将功能分支合并到develop分支：

```
git checkout develop  
git merge --no-ff feature-x
```

删除feature分支：

```
git branch -d feature-x
```

git flow

预发布 (release) 分支

发布正式版本之前（即合并到master分支之前），可能需要有一个预发布的版本进行测试。

预发布分支是从develop分支上面分出来的

预发布结束以后，必须合并进develop和master分支

分支命名，可以采用release-*的形式

git flow

预发布 (release) 分支

创建一个预发布分支:

```
git checkout -b release-1.2 develop
```

确认没有问题后, 合并到master分支:

```
git checkout master  
git merge --no-ff release-1.2  
# 对合并生成的新节点, 做一个标签  
git tag -a 1.2
```

再合并到develop分支:

```
git checkout develop  
git merge --no-ff release-1.2
```

最后, 删除预发布分支: `git branch -d release-1.2`

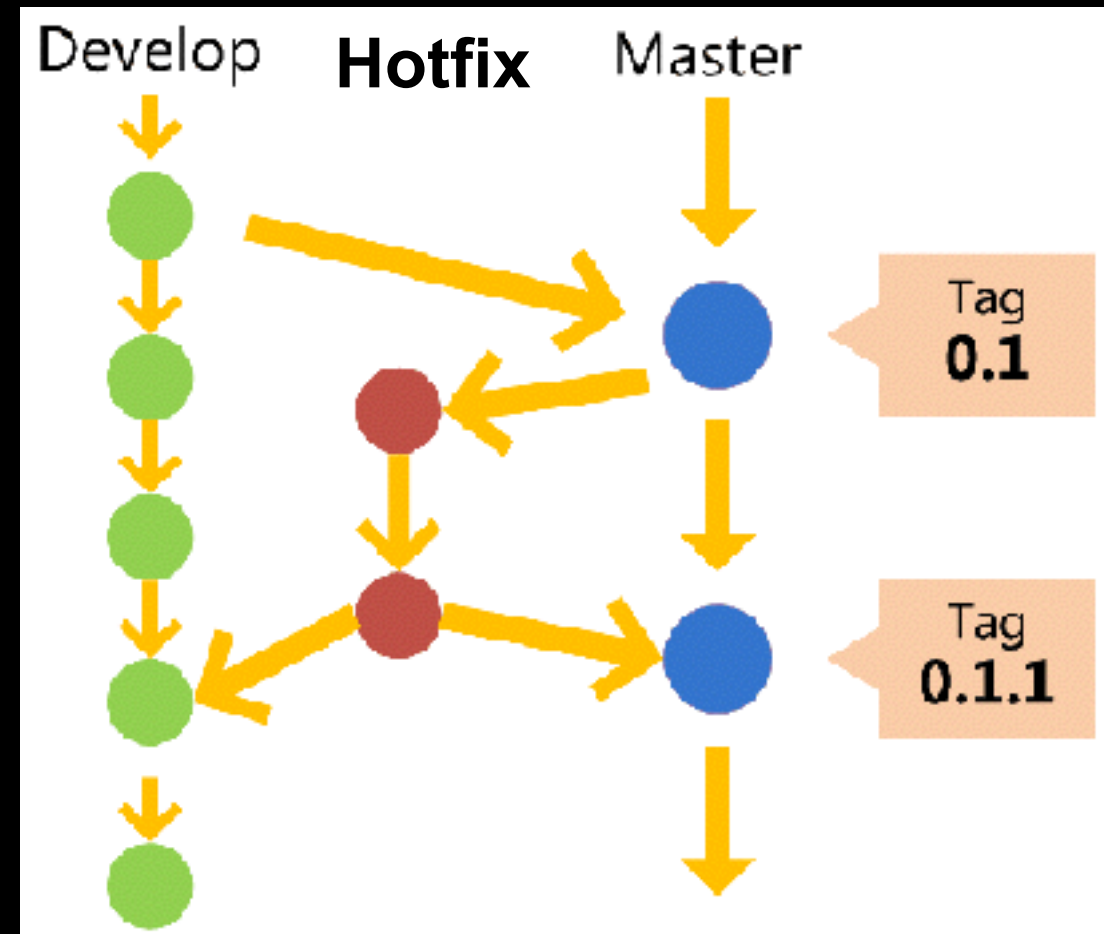
git flow

修补bug (hotfix) 分支

修补bug分支是从master分支上面分出来的

修补结束以后，再合并进master和develop分支

分支命名，可以采用hotfix-*的形式



git flow

修补bug (hotfix) 分支

创建一个修补bug分支：

```
git checkout -b hotfix-0.1 master
```

修补结束后，合并到master分支：

```
git checkout master  
git merge --no-ff hotfix-0.1  
git tag -a 0.1.1
```

再合并到develop分支：

```
git checkout develop  
git merge --no-ff hotfix-0.1
```

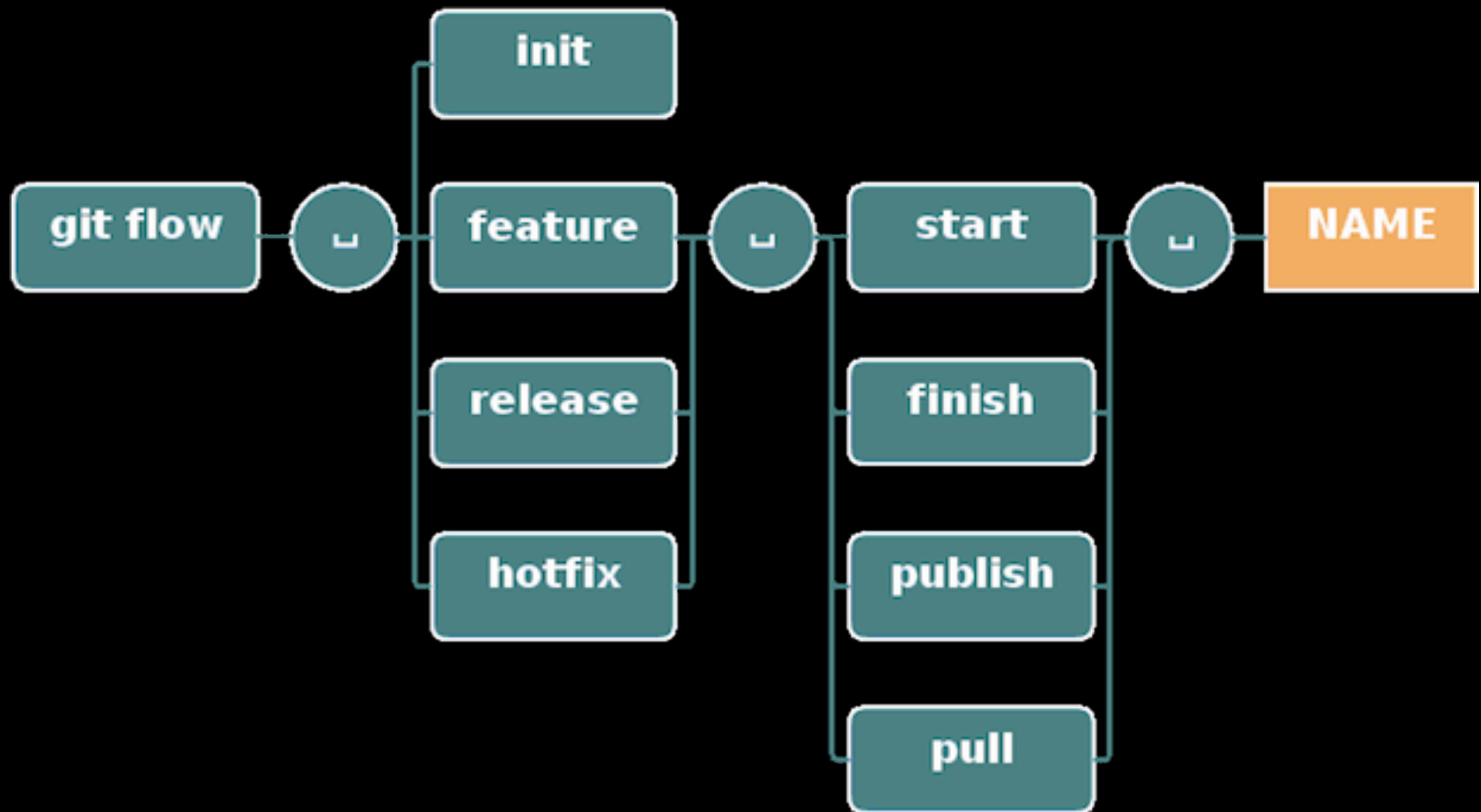
最后，删除"修补bug分支"：

```
git branch -d hotfix-0.1
```

git flow流程图



git-flow 工具



<https://github.com/nvie/gitflow>

git-flow工具

- 初始化: `git flow init`
- 开始新Feature: `git flow feature start FEATURE-*`
- Publish一个Feature(push到远程): `git flow feature publish FEATURE-*`
- 获取Publish的Feature: `git flow feature pull origin FEATURE-*`
- 完成一个Feature: `git flow feature finish FEATURE-*`
- 开始一个Release: `git flow release start RELEASE [BASE]`
- Publish一个Release: `git flow release publish RELEASE`
- 发布Release: `git flow release finish RELEASE` 别忘了 `git push -tags`
- 开始一个Hotfix: `git flow hotfix start VERSION [BASENAME]`
- 发布一个Hotfix: `git flow hotfix finish VERSION`

<https://github.com/nvie/gitflow>

git flow GUI



SourceTree

The screenshot displays the SourceTree application interface for a repository named 'webpy (Git)'. The top toolbar includes icons for Commit, Stash, Refresh, Pull, Fetch, Push, Branch, Merge, Tag, and Git Flow. The left sidebar shows the 'WORKSPACE' with sections for File status, History, Search, BRANCHES (listing master, HEAD, python3, webpy-0.22, webpy-0.23), TAGS, REMOTES (listing origin), STASHES, SUBMODULES, and SUBTREES.

The main area is divided into three panes. The top pane shows a commit history table with columns for Graph, Description, Commit, Author, and Date. The bottom-left pane shows the selected commit's details, including the commit hash, parents, author, date, and labels. The bottom-right pane shows a diff view for the file 'web/webapi.py', highlighting changes in lines 8-30 and 301-307.

Graph	Description	Commit	Author	Date
[Graph icon]	Merge pull request #351 from soneyard/mast...	770bef8	Anand Chitip...	Apr 6, 2016, 2:40...
[Graph icon]	Include Error Code '451 Unavailable For Legal Reasons'	632a8b0	Yannik Robin K...	Dec 23, 2015, 3:43...
[Graph icon]	Merge pull request #361 from jzellman/tihurl-simple-case	18323a2	Anand Chitipot...	Mar 19, 2015, 9:32 A...
[Graph icon]	Change dburl2dict to use urlparse and to support the simple case of just a database name.	905a88b	Jeff Zelman <j...	Mar 10, 2015, 10:29...
[Graph icon]	Merge pull request #254 from jzellman/master	a27befb	Anand Chitipot...	Jan 14, 2016, 2:30 AM
[Graph icon]	Escape HTML characters when emitting documentation.	c1e739c	Jeff Zelman <j...	Oct 24, 2013, 11:08...
[Graph icon]	Merge pull request #346 from goodrone/patch-1	503d560	Anand Chitipot...	Oct 15, 2015, 12:14...
[Graph icon]	Update templating.rst to remove external link	88f4a3a	goodrone <gp...	Oct 15, 2015, 3:55 AM
[Graph icon]	python 2.5 is no more supported by travis.	7f629f4	Anand Chitipot...	Mar 16, 2015, 2:40 A...
[Graph icon]	Fixed error in validip0addr on Windows.	25a5417	Anand Chitipot...	Mar 16, 2015, 2:39 AM
[Graph icon]	Merge pull request #324 from suhashpatil/patch-1	f48d17e	Anand Chitipot...	Mar 16, 2015, 1:49 AM
[Graph icon]	fix for https://github.com/webpy/webpy/issues/323	dcc0173	suhashpatil <s...	Mar 15, 2015, 12:57...

Selected commit details:

- Commit: 770baf8c686ba8de37333325...
- Parents: 18323a2799, 632a8b05eb
- Author: Anand Chitipothu <anandolog...
- Date: April 6, 2016 at 2:40:01 AM G...
- Labels: HEAD -> master origin/maste...

Diff view for web/webapi.py:

Hunk 1: Lines 8-30

```
8 8      "header", "debug",
9 9      "input", "data",
10 10     "setcookie", "cookies",
11 -     "ctx",
12 -     "HTTPError",
11 +     "ctx",
12 +     "HTTPError",
13 13
14 14     # 200, 201, 202, 204
15 -     "OK", "Created", "Accepted", "NoContent",
15 +     "OK", "Created", "Accepted", "NoContent",
16 16     "ok", "created", "accepted", "nocontent",
17 -
17 +
18 18     # 301, 302, 303, 304, 307
```

git flow GUI



1. 用git-flow初始化



Initialising repository for: git-flow

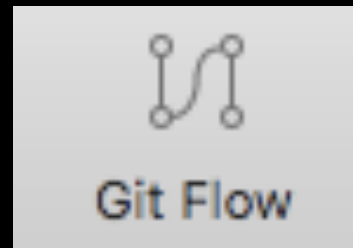
Create / use the following branches:

Production branch:	<input type="text" value="master"/>
Development branch:	<input type="text" value="develop"/>

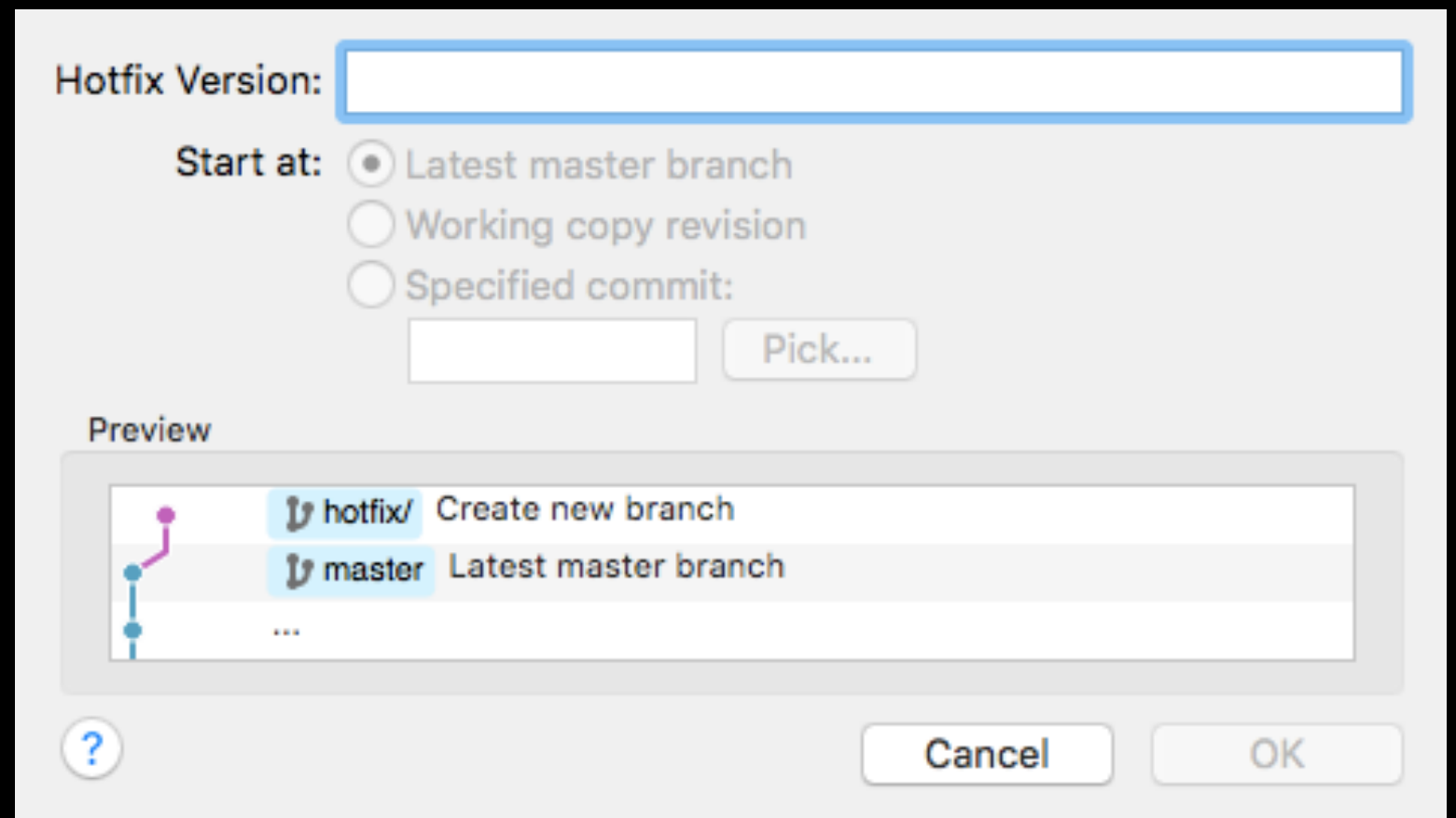
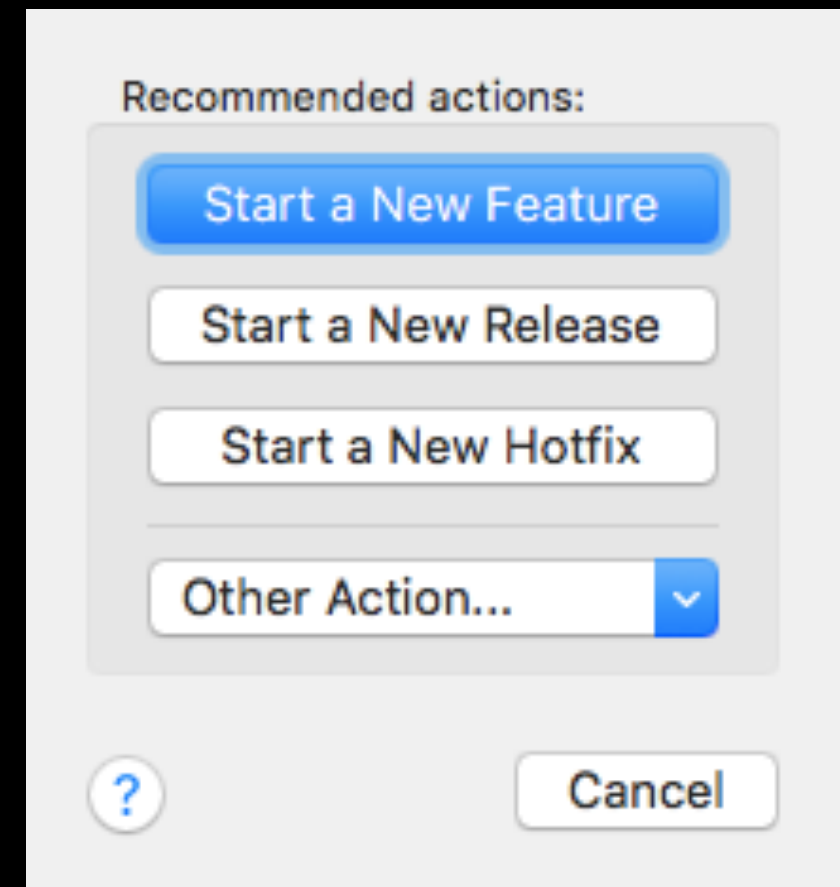
Use the following prefixes in future:

Feature branch prefix:	<input type="text" value="feature/"/>
Release branch prefix:	<input type="text" value="release/"/>
Hotfix branch prefix:	<input type="text" value="hotfix/"/>
Version tag prefix:	<input type="text"/>

git flow GUI



2. 点击git flow菜单，选择start new feature, release或者hotfix



git flow GUI



3. 做完后再次选择git flow菜单，
点击Finish
Action

