手動 git flow

1. 主要分支

在遠程倉庫中有兩個主要分支的生命期可以無限長，分別是： **Master Develop**

> **master** 分支（**origin/master**）

代碼倉庫中有且僅有的一條主分支，默認為 master ，在創建版本庫時會自動創建。所有提供給用戶使用的正式版本的源碼，都會在這個分支上發佈。也就是說主分支 master 用來發佈重大版本。

> **develop** 分支（**origin/develop**）

日常開發工作都會在 develop 分支上面完成。develop 分支可以用來生成代碼的最新隔夜版本（nightly builds）。

> 創建 **develop** 分支

```
$ git checkout -b develop master

#push develop 到遠程倉庫
$ git push origin develop
```

當我們在 `develop` 上完成了新版本的功能，最終會把所有的修改 `merge` 到 **master** 分支。針對每次 **master** 的修改都會打一個 Tag 作為可發佈產品的版本號。

2. 輔助分支

開發過程中不可能項目人所有都在一個 `develop` 分支中開發，版本管理會很混亂。所以除了主要分支外，我們還需要一些輔助分支來協助團隊成員間的並行開發。

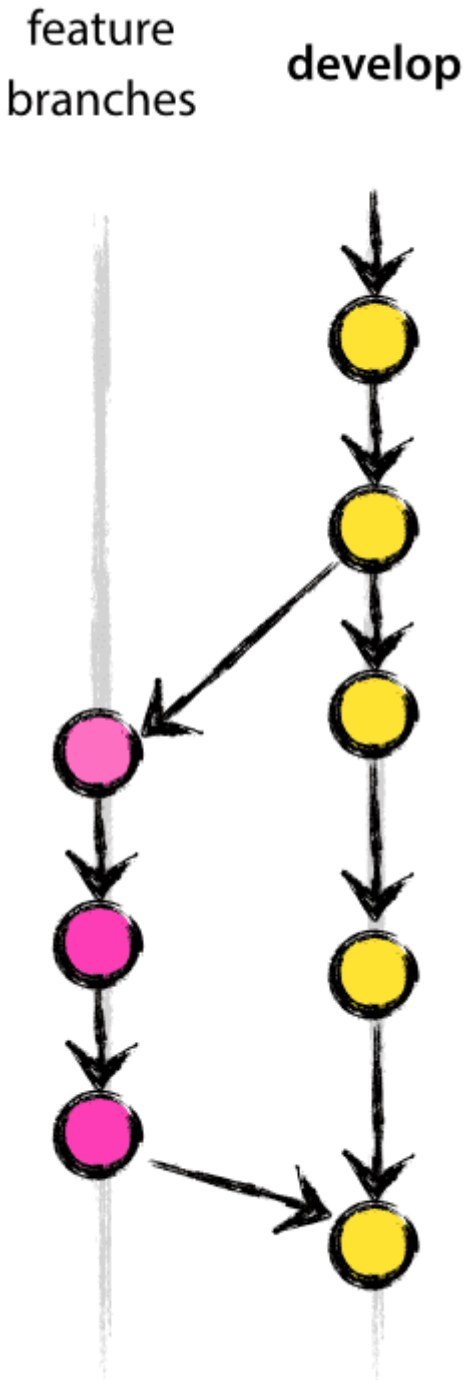
所用到的輔助分支大體分三類：

- **Feature branches**（功能分支）
- **Release branches**（預發佈分支）
- **Hotfix branches**（熱修復分支）

通過分支名我們能知道各類型分支都有特定作用，對於他們各自的起始分支和最終的合併分支也都有嚴格規定。呼，雖然可能會麻煩點，但讓人一目瞭然的效果還是很誘人的。

下面逐一介紹下各類型分支的創建使用和移除方法，過程中我在 `Github` 中創建一個虛擬的項目用來熟悉整個流程，或許你也可以像我一樣做一遍。哈，動手總會有意外收穫嘛。廢話少說，繼續正題～

2.1. Feature branches（功能分支）



> 應用場景：

當要開始一個新功能的開發時，我們可以創建一個 `Feature branche`。等待這個新功能開發完成並確定應用到新版本中就合併回 `develop`，那麼如果不是就會被很遺憾的丟棄。。。

> 應用規則：

1. 從 `develop` 分支創建，最終合併回 `develop` 分支;
2. 分支名：feature / *;

Tips：這裡很多地方說用 **feature-*** 的方式命名，因為公司項目中用的 **feature / *** 方式，也就習慣了，其實意思是一樣的。

(1). Creat a feature branch

```
$ git checkout -b feature/test develop
```

do something in `feature/test` branch

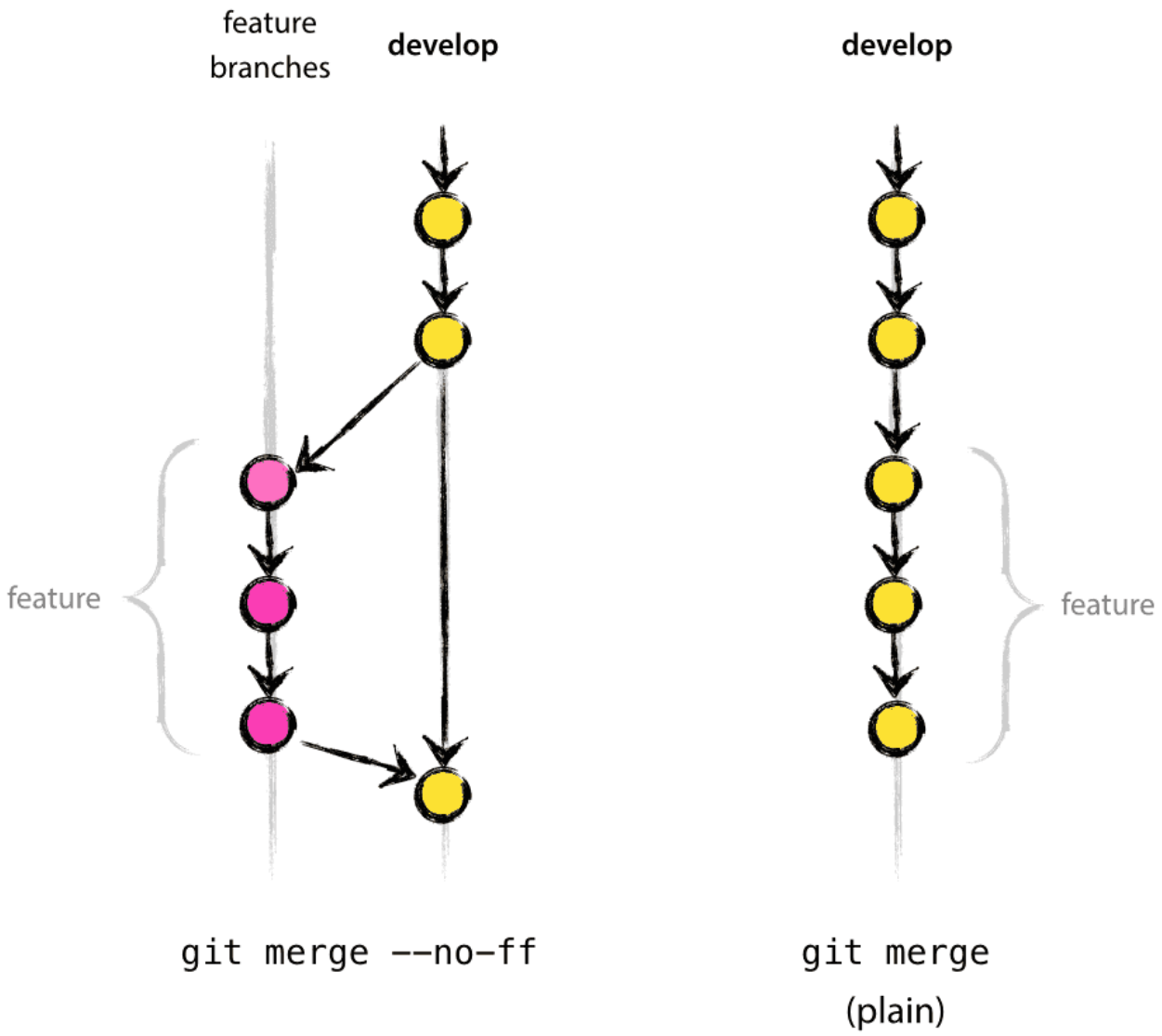
push 本地 `feature/test` 到遠處代碼庫:

```
$ git push origin feature/test
```

(2). 切換到 **develop** 合併 **feature/test**

```
$ git checkout develop

$ git merge --no-ff feature/test
```

「- -no-ff」 的作用是創建一個新的 「commit」 對象用於當前合併操作。這樣既可以避免丟失該功能分支的歷史存在信息，又可以集中該功能分支所有歷史提交。並且如果想回退版本也會比較方便。



(3). 移除本地和遠程倉庫的 **feature/test** 分支

```
$ git branch -d feature/test

$ git push origin --delete feature/test
```

2.2. Release branches（預發佈分支）

> 應用場景：

「Release branches」 用來做新版本發佈前的準備工作，在上面可以做一些小的 bug 修復、準備發佈版本號等等和發佈有關的小改動，其實已經是一個比較成熟的版本了。另外這樣我們既可以在預發佈分支上做一些發佈前準備，也不會影響 「develop」 分支上下一版本的新功能開發。

> 應用規則：

1. 從 `develop` 分支創建，最終合併回 `develop` 和 `master`；
2. 分支名：release-*；

(1). Creat a release branch

```
$ git checkout -b release-1.1 develop

#push 到遠程倉庫（可選）
$ git push origin release-1.1
```

do something in `release-1.1` branch

(2). 切換到 **master** 合併 **release-1.1**

```
$ git checkout master

$ git merge --no-ff release-1.1

$ git tag -a 1.1

$ git push origin 1.1
```

當我們的 `release-1.1` 的 Review 完成，也就預示著我們可以發佈了。打上相應的版本號，再 **push** 到遠程倉庫。

(3). 切換到 **develop** 合併 **release-1.1**

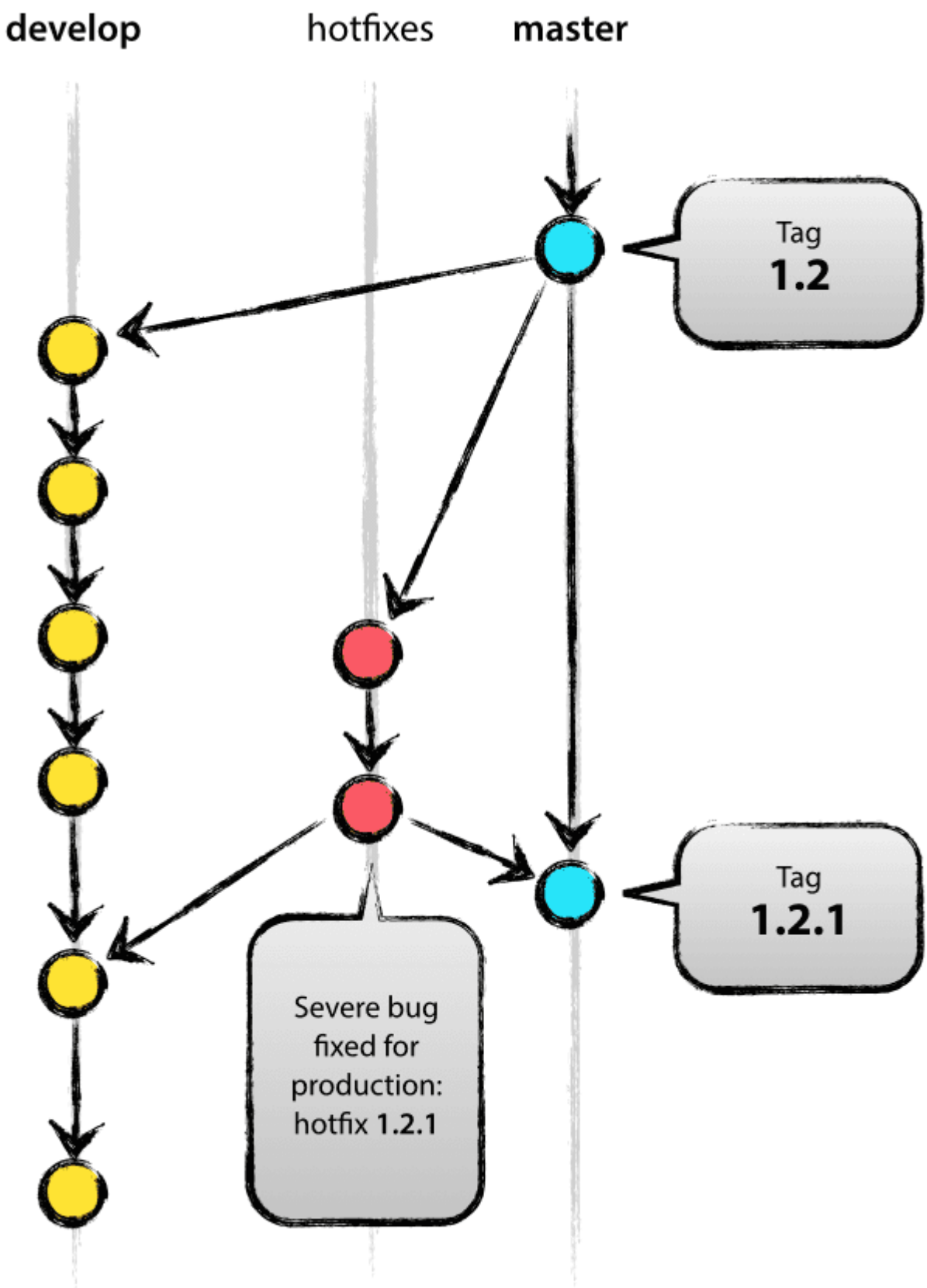
預發佈分支所做的修改同時也要合併回 **develop**

```
$ git checkout develop

$ git merge --no-ff release-1.1
```

(4). 移除本地和遠程倉庫的 **release-1.1**

```
$ git branch -d release-1.1

$ git push origin --delete release-1.1
```

2.3. Hotfix branches（熱修復分支）



> 應用場景：

「Hotfix branches」 主要用於處理線上版本出現的一些需要立刻修復的 bug 情況.

> 應用規則：

1. 從 `master` 分支上當前版本號的 `tag` 處切出，也就是從最新的 `master` 上創建，最終合併回 `develop` 和 `master`；
2. 分支名：hotfix-＊；

(1). Creat a fixbug branch

```
$ git checkout -b fixbug-1.1.1 master

#push 到遠程倉庫（可選）
$ git push origin fixbug-1.1.1
```

do something in `fixbug-1.1.1` branch

(2). 切換到 master 合併 fixbug-1.1.1

```
$ git checkout master

$ git merge --no-ff fixbug-1.1.1

$ git tag -a 1.1.1

$ git push origin 1.1.1`
```

bug 修復完成，合併回 `master` 並打上版本號；

(3). 切換到 develop 合併 fixbug－1.1.1

```
$ git checkout develop

$ git merge --no-ff fixbug-1.1.1
```

(4). 移除本地和遠程倉庫的 fixbug-1.1.1

```
$ git branch -d fixbug-1.1.1

$ git push origin --delete fixbug-1.1.1
```