

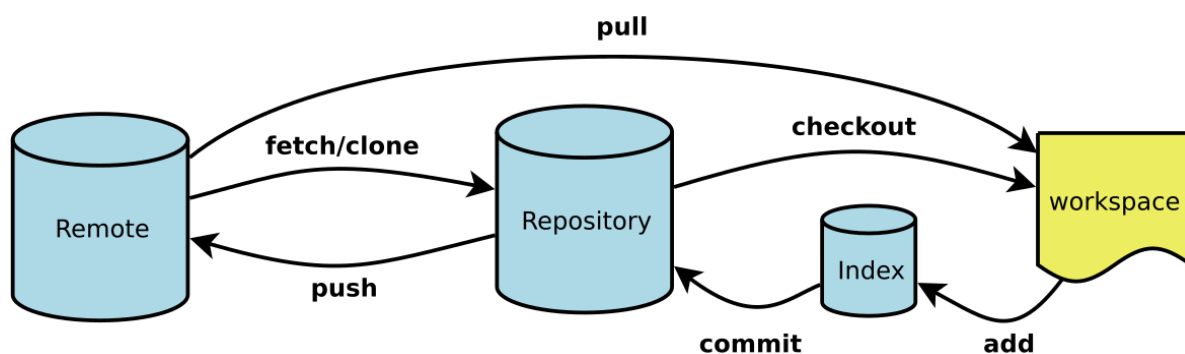
Git 笔记

- <https://zhuanlan.zhihu.com/p/30044692>
- <https://help.gitee.com/enterprise>
- https://mp.weixin.qq.com/s?__biz=Mzg2OTg3MTU2OQ==&mid=2247506565&idx=1&sn=3a332a80999b45008d977fb6a10a2275&source=41#wechat_redirect
- <https://help.gitee.com/questions/GitHub%E4%BB%93%E5%BA%93%E5%BF%AB%E9%80%9F%E5%AF%BC%E5%85%A5Gitee%E5%8F%8A%E5%90%8C%E6%AD%A5%E6%9B%B4%E6%96%B0#article-header1>

Git 笔记

1. Git 工作流程
2. Git 分支操作
 - 2.1 分支基本操作
 - 2.2 主分支Master
 - 2.3 开发分支Develop
 - 2.3.1 操作
 - 2.4 临时性分支
 - 2.4.1 功能分支
 - 2.4.1.1 操作
 - 2.4.2 预发布分支
 - 2.4.2.1 操作
 - 2.4.3 修补bug分支
 - 2.4.3.1 操作
 - 2.5 保存当前工作创建新分支
 - 2.6 综合示例
3. Git 基础操作
 - 3.1 Git配置
 - 3.2 仓库基本管理
 - 3.3 版本回退
 - 3.4 撤销修改和删除文件操作。
4. Git远程仓库管理
 - 4.1 修改仓库名
 - 4.2 添加一个仓库
 - 4.3 查看当前仓库对应的远程仓库地址
 - 4.4 修改仓库对应的远程仓库地址
 - 4.5 抓取分支
5. 冲突解决

1. Git 工作流程



Workspace: 工作区

Index / Stage: 暂存区

Repository: 仓库区 (或本地仓库)

Remote: 远程仓库

2. Git 分支操作

2.1 分支基本操作

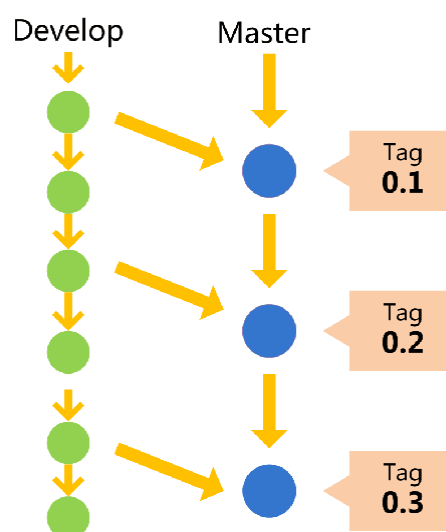
1. `git checkout -b dev` 创建并切换到"dev"分支上
2. `git branch` 查看分支，会列出所有的分支，当前分支前面会添加一个星号
 1. 列出所有本地分支
`$ git branch`
 2. 列出所有远程分支
`$ git branch -r`
 3. 列出所有本地分支和远程分支
`$ git branch -a`
3. `git checkout dev` 切换到"dev"分支上
4. `git merge dev` 用于合并指定"dev"分支到当前分支上
5. `git branch -d dev` 删除"dev"分支

2.2 主分支Master

- 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。

2.3 开发分支Develop

- 开发用的分支，叫做Develop。
- 这个分支可以用来生成代码的最新隔夜版本（nightly）。如果想正式对外发布，就在**Master分支上，对Develop分支进行"合并"（merge）**。



2.3.1 操作

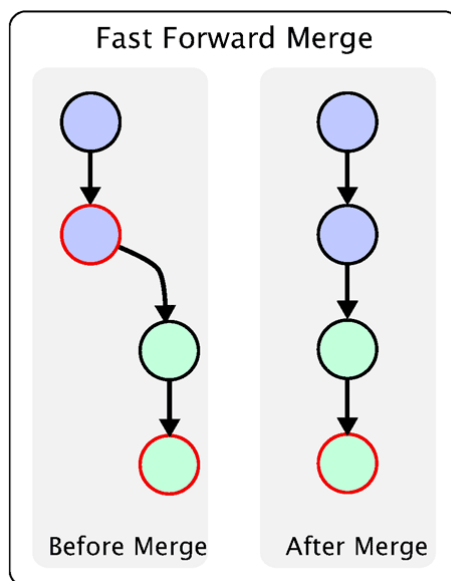
Git创建Develop分支的命令：

```
1 | git checkout -b develop master
```

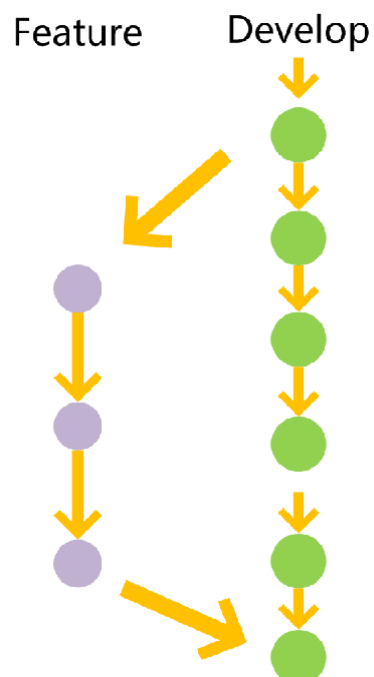
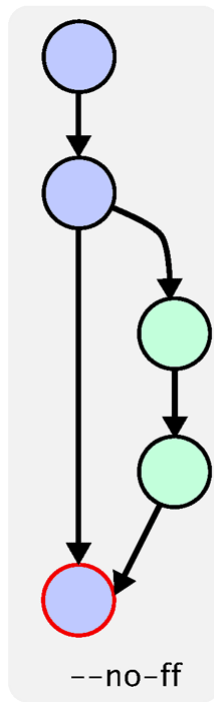
将Develop分支发布到Master分支的命令：

```
1 | # 切换到Master分支  
2 | git checkout master  
3 |  
4 | # 对Develop分支进行合并  
5 | git merge --no-ff develop
```

命令的 `--no-ff` 参数是禁用快进式合并。默认情况下，Git执行"快进式合并"（fast-forward merge），会直接将Master分支指向Develop分支。



使用`--no-ff`参数后，会执行正常合并，在Master分支上生成一个新节点。为了保证版本演进的清晰，我们希望采用这种做法。



2.4.1.1 操作

功能分支的名字，可以采用feature-*的形式命名。

创建一个功能分支：

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

开发完成后，将功能分支合并到develop分支：

```
1 | git checkout develop
2 |
3 | git merge --no-ff feature-x
```

删除feature分支：

```
1 | git branch -d feature-x
```

2.4.2 预发布分支

- 预发布分支是从Develop分支上面分出来的，发布正式版本之前（即合并到Master分支之前），我们可能需要有一个预发布的版本进行测试。
- 预发布结束以后，必须合并进Develop和Master分支

2.4.2.1 操作

它的命名，可以采用release-*的形式。

创建一个预发布分支：

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

确认没有问题后，合并到master分支：

```
1 | git checkout master
2 |
3 | git merge --no-ff release-1.2
4 |
5 | # 对合并生成的新节点，做一个标签
6 | git tag -a 1.2
```

再合并到develop分支：

```
1 | git checkout develop
2 |
3 | git merge --no-ff release-1.2
```

最后，删除预发布分支：

```
1 | git branch -d release-1.2
```

2.4.3 修补bug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。

2.4.3.1 操作

创建一个修补bug分支：

```
1 | git checkout -b fixbug-0.1 master
```

修补结束后，合并到master分支：

```
1 | git checkout master
2 |
3 | git merge --no-ff fixbug-0.1
4 |
5 | git tag -a 0.1.1
```

再合并到develop分支：

```
1 | git checkout develop
2 |
3 | git merge --no-ff fixbug-0.1
```

最后，删除"修补bug分支"：

```
1 | git branch -d fixbug-0.1
```

2.5 保存当前工作创建新分支

比如我在开发中接到一个404 bug时候，我们可以创建一个404分支来修复它，但是，当前的"dev"分支上的工作还没有提交。Git提供了一个stash功能，可以把当前工作现场"隐藏起来"，等以后恢复现场后继续工作。

1. 将当前的工作现场隐藏起来

```
1 | git stash
```

2. 恢复工作现场

```
1 | #恢复工作现场,恢复后，stash内容并不删除，你需要使用命令git stash drop来删除。
2 | git stash apply
3 |
4 | #恢复的同时把stash内容也删除了
5 | git stash pop
```

3. 删除工作现场

```
1 | git stash drop
```

2.6 综合示例

```
E73-8@E73-8-PC /d/www/testGit (dev)
$ git stash
Saved working directory and index state WIP on dev: 91dfe16 merge with no-ff
HEAD is now at 91dfe16 merge with no-ff

E73-8@E73-8-PC /d/www/testGit (dev)
$ git status
On branch dev
nothing to commit, working directory clean

E73-8@E73-8-PC /d/www/testGit (dev)
$
```

将当前的工作现场隐藏起来

查看状态，是干净的

通过创建issue-404分支来修复bug了。

首先我们要确定在那个分支上修复bug，比如我现在是在主分支master上来修复的，现在我要在master分支上创建一个临时分支，演示如下：

```
E73-8@E73-8-PC /d/www/testGit (master)
$ git checkout -b issue-404
Switched to a new branch 'issue-404'

E73-8@E73-8-PC /d/www/testGit (issue-404)
$ cat readme.txt
1111111111111111
2222222222222222
3333333333333333
4444444444444444
6666666666666666
7777777777777777
9999999999999999
0101010101010101
bbbbbbbbbbbbbbbb

E73-8@E73-8-PC /d/www/testGit (issue-404)
$ cat readme.txt
1111111111111111
2222222222222222
3333333333333333
4444444444444444
6666666666666666
7777777777777777
9999999999999999
0101010101010101
aaaaaaaaaaaaaaaa

E73-8@E73-8-PC /d/www/testGit (issue-404)
$ git add readme.txt

E73-8@E73-8-PC /d/www/testGit (issue-404)
$ git commit -m "fix bug 404"
[issue-404 5198735] fix bug 404
1 file changed, 1 insertion(+), 1 deletion(-)

E73-8@E73-8-PC /d/www/testGit (issue-404)
$
```

在master分支上创建临时分支issue-404

未修改前查看readme.txt内容

修改后把readme.txt内容最后一行bbbbbb改成aaaaaa

修复完成后，切换到master分支上，并完成合并，最后删除issue-404分支。演示如下：

```
E73-8@E73-8-PC /d/www/testGit (issue-404)
$ git checkout master
Switched to branch 'master'
Your branch is ahead of 'origin/master' by 8 commits.
(use "git push" to publish your local commits)

E73-8@E73-8-PC /d/www/testGit (master)
$ git merge --no-ff -m "merge bug fix 404" issue-404
Merge made by the 'recursive' strategy.
 readme.txt | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

E73-8@E73-8-PC /d/www/testGit (master)
$ cat readme.txt
1111111111111111
2222222222222222
3333333333333333
4444444444444444
6666666666666666
7777777777777777
9999999999999999
0101010101010101
aaaaaaaaaaaaaaaa
```

切换到master分支上

合并分支issue-404内容

合并分支后查看内容如下，和issue-404内容一致

在master分支上删除临时分支 issue-404

现在，我们回到dev分支上干活了。

```
E73-8@E73-8-PC /d/www/testGit (master)
$ git checkout dev
Switched to branch 'dev'

E73-8@E73-8-PC /d/www/testGit (dev)
$ git status
On branch dev
nothing to commit, working directory clean
```

从master分支切换到dev分支上

目前干净的

工作区是干净的，那么我们工作现场去哪里呢？我们可以使用命令 `git stash list` 来查看下。如下：

```
E73-8@E73-8-PC /d/www/testGit (dev)
$ git stash list
stash@{0}: WIP on dev: 91dfe16 merge with no-ff
stash@{1}: WIP on master: 91dfe16 merge with no-ff
stash@{2}: WIP on master: 91dfe16 merge with no-ff
```

工作现场还在，Git把stash内容存在某个地方了，但是需要恢复一下，可以使用如下2个方法：

- 1.git stash apply恢复，恢复后，stash内容并不删除，你需要使用命令git stash drop来删除。
- 2.另一种方式是使用git stash pop,恢复的同时把stash内容也删除了。

演示如下


```
E73-8@E73-8-PC /d/www/testGit (dev)
$ git stash list
stash@{0}: WIP on dev: 91dfe16 merge with no-ff
stash@{1}: WIP on master: 91dfe16 merge with no-ff
stash@{2}: WIP on master: 91dfe16 merge with no-ff
E73-8@E73-8-PC /d/www/testGit (dev)
$ git stash drop
Dropped refs/stash@{0} (d228a8c52dcf5d89aec877f0c9be774a73eb8a34)
E73-8@E73-8-PC /d/www/testGit (dev)
$ git stash list
stash@{0}: WIP on master: 91dfe16 merge with no-ff
stash@{1}: WIP on master: 91dfe16 merge with no-ff
E73-8@E73-8-PC /d/www/testGit (dev)
$ git stash list
stash@{0}: WIP on master: 91dfe16 merge with no-ff
stash@{1}: WIP on master: 91dfe16 merge with no-ff
E73-8@E73-8-PC /d/www/testGit (dev)
$ git stash drop
Dropped refs/stash@{0} (683d3fe8c3416d95e8dd25d3055a5b0f376d8f0c)
E73-8@E73-8-PC /d/www/testGit (dev)
$ git stash drop
Dropped refs/stash@{0} (753a3b3dad781bab43560494b836bab1860cd5e)
E73-8@E73-8-PC /d/www/testGit (dev)
$ git stash list
E73-8@E73-8-PC /d/www/testGit (dev)
```

删除前

删除一条

剩下2条

继续删2条

没有了

3. Git 基础操作

3.1 Git配置

1. 显示当前的Git配置

```
1 | git config --list
```

2. 编辑Git配置文件

```
1 | git config -e [--global] # 设置提交代码时的用户信息
2 | git config [--global] user.name "[name]"
3 | git config [--global] user.email "[email address]"
```

3.2 仓库基本管理

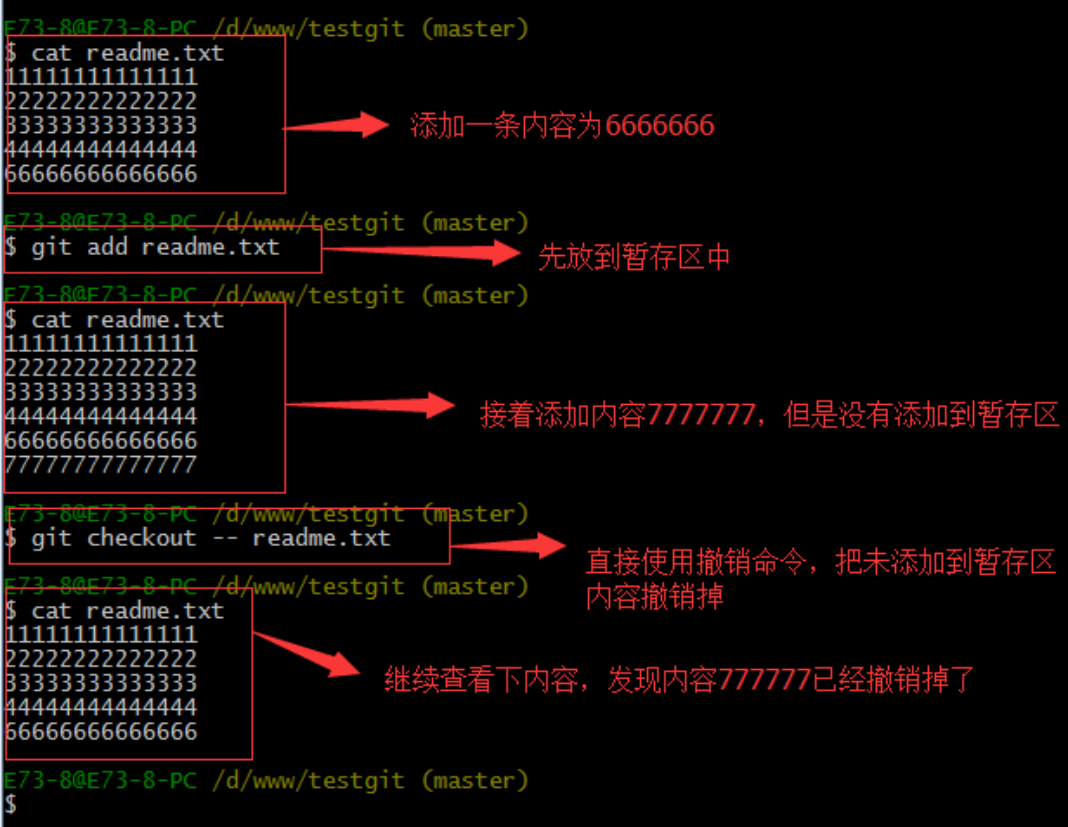
1. `git init` 把这个目录变成git可以管理的仓库
2. `git clone [url]` 下载一个项目
3. `git add readme.txt` 添加 `readme.txt` 到暂存区里面去
 1. `git add -A` 或 `git add .` 可以提交当前仓库的所有改动。
4. `git commit -m "提交信息"` 从Git的暂存区提交版本到仓库，参数-m后为当次提交的备注信息
5. `git status -s` 来查看是否还有文件未提交
6. `git diff readme.txt` 查看readme.txt文件到底改了什么内容
7. `git log` 查看git提交的日志
 1. `git log --pretty=oneline` 简洁显示

3.3 版本回退

1. `git reset --hard HEAD^` 回退到上个版本
2. 那么如果要回退到上上个版本只需把 `HEAD^` 改成 `HEAD^^` 以此类推。
3. `git reset --hard HEAD~100` 回退到前100个版本
4. `git reset --hard 版本号` 版本号由 `git reflog` 获得
5. `git reset --hard origin/master` 回退到和远程版本一样,origin代表你远程仓库的名字, master代表分支名

3.4 撤销修改和删除文件操作。

1. `git reset --hard HEAD^` 直接恢复到上一个版本
2. `git checkout -- readme.txt` 把readme.txt文件在工作区做的修改全部撤销.这里有2种情况, 如下:
 1. readme.txt自动修改后, 还没有放到暂存区, 使用 撤销修改就回到和版本库一模一样的状态。
 2. 另外一种readme.txt已经放入暂存区了, 接着又作了修改, 撤销修改就回到添加暂存区后的状态。



The terminal screenshot shows a series of commands and file contents with red arrows pointing to specific actions:

- `E73-8@E73-8-PC /d/www/testgit (master)`
`$ cat readme.txt`
11111111111111
22222222222222
33333333333333
44444444444444
66666666666666
→ 添加一条内容为6666666
- `E73-8@E73-8-PC /d/www/testgit (master)`
`$ git add readme.txt`
→ 先放到暂存区中
- `E73-8@E73-8-PC /d/www/testgit (master)`
`$ cat readme.txt`
11111111111111
22222222222222
33333333333333
44444444444444
66666666666666
77777777777777
→ 接着添加内容7777777, 但是没有添加到暂存区
- `E73-8@E73-8-PC /d/www/testgit (master)`
`$ git checkout -- readme.txt`
→ 直接使用撤销命令, 把未添加到暂存区内容撤销掉
- `E73-8@E73-8-PC /d/www/testgit (master)`
`$ cat readme.txt`
11111111111111
22222222222222
33333333333333
44444444444444
66666666666666
→ 继续查看下内容, 发现内容777777已经撤销掉了
- `E73-8@E73-8-PC /d/www/testgit (master)`
`$`

3. `rm b.txt` 可以直接在文件目录中把文件删了.如果我想彻底从版本库中删掉了此文件的话, 可以再执行 `commit`命令 提交掉
4. `git rm [file1] [file2] ...` 停止追踪指定文件, 但该文件会保留在工作区
5. `git checkout -- b.txt` 恢复b.txt文件(没有commit之前)

4. Git远程仓库管理

4.1 修改仓库名

一般来讲，默认情况下，在执行clone或者其他操作时，仓库名都是 `origin` 如果说我们想给他改改名字，比如我不喜欢origin这个名字，想改为 `oschina` 那么就要在仓库目录下执行命令：

```
1 | git remote rename origin oschina
```

这样 你的远程仓库名字就改成了oschina，同样，以后推送时执行的命令就不再是 `git push origin master` 而是 `git push oschina master` 拉取也是一样的

4.2 添加一个仓库

在不执行克隆操作时，如果想将一个远程仓库添加到本地的仓库中，可以执行

```
1 | git remote add origin 仓库地址
```

注意：

1. origin是你的仓库的别名 可以随便改，但请务必不要与已有的仓库别名冲突
2. 仓库地址一般来讲支持 http/https/ssh/git协议，其他协议地址请勿添加

4.3 查看当前仓库对应的远程仓库地址

```
1 | git remote -v
```

这条命令能显示你当前仓库中已经添加了的仓库名和对应的仓库地址，通常来讲，会有两条一模一样的记录，分别是fetch和push，其中fetch是用来从远程同步 push是用来推送到远程

4.4 修改仓库对应的远程仓库地址

```
1 | git remote set-url origin 仓库地址
```

4.5 抓取分支

1. 现在要在dev分支上做开发，就必须把远程的origin的dev分支到本地来，于是可以使用命令创建本地dev分支：

```
1 | #创建本地dev分支
2 | git checkout -b dev origin/dev
3 |
4 | #把现在的dev分支推送到远程去
5 | git push origin dev
```

2. 当需要把最新的提交从origin/dev抓下来

```
1 | git pull 远程库名 分支名
2 | eg: git pull origin dev
```

- 一次性拉取该仓库的所有分支

```
1 | $ for b in $(git branch -r | grep -v -- '->'); do git branch --track
    ${b##origin/} $b; done
```

命令简单解释：

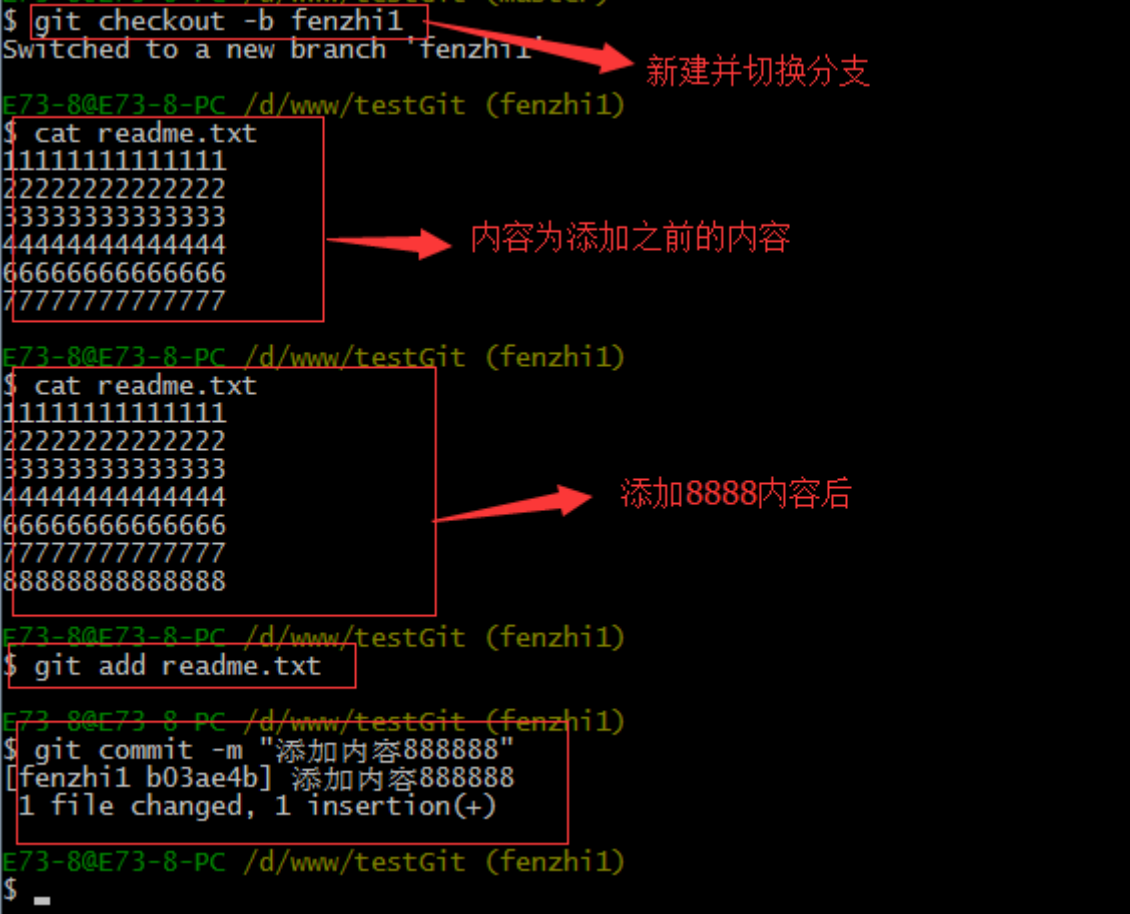
- | 表示通道，即前面命令的输出是后面命令的输入。
- for xxx in xxxs; do xxx; done 是 shell 的 for 循环语句。
- 倒引号 `` 表示里面的命令。
- git branch -r 列出远程分支。
- grep -v -'>' , grep 查找命令, -v 参数表示 not grep, 即查找输入中的不含'>'的行。
- git branch -r | grep -v -'>' , 合起来就是参看远程分支中除了含有'>'的分支。
- \$b 表示远程分支名, 例如: origin/dev。
- \${b##origin/} 表示截取远程分支名中 origin/后面的内容, 例如: dev, 以此当做本地分支。
- git branch --track \${b##origin/} \$b, 类似于 git branch dev origin/dev, -track 参数是默认的, 不加亦可。

3. 指定本地dev分支与远程origin/dev分支的链接

```
1 | git branch --set-upstream dev origin/dev
```

5. 冲突解决

下面我们还是一步一步来，先新建一个新分支，比如名字叫fenzhi1，在readme.txt添加一行内容8888888，然后提交，如下所示：



```
$ git checkout -b fenzhi1
Switched to a new branch 'fenzhi1'

E73-8@E73-8-PC /d/www/testGit (fenzhi1)
$ cat readme.txt
1111111111111111
2222222222222222
3333333333333333
4444444444444444
6666666666666666
7777777777777777

E73-8@E73-8-PC /d/www/testGit (fenzhi1)
$ cat readme.txt
1111111111111111
2222222222222222
3333333333333333
4444444444444444
6666666666666666
7777777777777777
8888888888888888

E73-8@E73-8-PC /d/www/testGit (fenzhi1)
$ git add readme.txt

E73-8@E73-8-PC /d/www/testGit (fenzhi1)
$ git commit -m "添加内容8888888"
[fenzhi1 b03ae4b] 添加内容8888888
1 file changed, 1 insertion(+)
```

新建并切换分支

内容为添加之前的内容

添加8888内容后

同样，我们现在切换到master分支上来，也在最后一行添加内容，内容为99999999，如下所示：

```
E73-8@E73-8-PC /d/www/testGit (fenzhi1)
$ git checkout master
Switched to branch 'master'
Your branch is ahead of 'origin/master' by 1 commit.
(use "git push" to publish your local commits)

E73-8@E73-8-PC /d/www/testGit (master)
$ cat readme.txt
1111111111111111
2222222222222222
3333333333333333
4444444444444444
6666666666666666
7777777777777777

E73-8@E73-8-PC /d/www/testGit (master)
$ cat readme.txt
1111111111111111
2222222222222222
3333333333333333
4444444444444444
6666666666666666
7777777777777777
9999999999999999

E73-8@E73-8-PC /d/www/testGit (master)
$ git add readme.txt

E73-8@E73-8-PC /d/www/testGit (master)
$ git commit -m "在master分支上新增内容99999"
master 418595b] 在master分支上新增内容99999
1 file changed, 1 insertion(+)

/d/www/testGit (master)

$ 3-8@E73-8-PC
```

切换到master分支上

未添加内容之前

添加内容9999之后

现在我们需要在master分支上来合并fenzhi1，如下操作：

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

E73-8@E73-8-PC /d/www/testGit (master|MERGING)
$ git status
On branch master
Your branch is ahead of 'origin/master' by 2 commits.
(use "git push" to publish your local commits)

You have unmerged paths.
(fix conflicts and run "git commit")

Unmerged paths:
  (use "git add <file>..." to mark resolution)

        both modified:   readme.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        testgit2/

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

E73-8@E73-8-PC /d/www/testGit (master|MERGING)
$ cat readme.txt
1111111111111111
2222222222222222
3333333333333333
4444444444444444
5666666666666666
7777777777777777
<==== HEAD
9999999999999999
=====
8888888888888888
>>>>>> fenzhil

E73-8@E73-8-PC /d/www/testGit (master|MERGING)
$
```

在master分支上合并fenzhil

产生冲突

查看状态

查看readme.txt内容

冲突代码

Git用<<<<<<, ==, >>>>>>标记出不同分支的内容, 其中<<<HEAD是指主分支修改的内容, >>>>>>fenzhil 是指fenzhil上修改的内容, 我们可以修改下如下后保存:

```
$ cat readme.txt
1111111111111111
2222222222222222
3333333333333333
4444444444444444
5666666666666666
7777777777777777
9999999999999999

E73-8@E73-8-PC /d/www/testGit (master|MERGING)
$ git add readme.txt

E73-8@E73-8-PC /d/www/testGit (master|MERGING)
$ git commit -m "conflict fixed"
[master 672c256] conflict fixed

E73-8@E73-8-PC /d/www/testGit (master)
$
```

查看内容, 修改成和主干上代码一样的

如果我想查看分支合并的情况的话, 需要使用命令 git log.命令行演示如下:

```
275-6@E75-6-PC /d/www/testgit (master)
$ git log
commit 672c25679deef1281775f0a800058ac1358234b8
Merge: 418595b b03ae4b
Author: longen0707 <879083421@qq.com>
Date: Mon Oct 20 18:01:16 2014 +0800
```

conflict fixed

```
commit 418595b4c39b47820cf00241a291505713a649e8
Author: longen0707 <879083421@qq.com>
Date: Mon Oct 20 17:52:47 2014 +0800
```

在master分支上新增内容99999

```
commit b03ae4b0c7fa088df21c097178ac3b3ad01dbee8
Author: longen0707 <879083421@qq.com>
Date: Mon Oct 20 17:48:42 2014 +0800
```

添加内容888888

```
commit 56ccde3b9b86b16c1dbb6670eaaf1f1f7ad40c06
Author: longen0707 <879083421@qq.com>
Date: Mon Oct 20 17:25:43 2014 +0800
```

dev分支上增加内容77777

```
commit 2a4fd81d920ba228941438f3262fe1ae60f1f5be
Author: longen0707 <879083421@qq.com>
Date: Mon Oct 20 14:57:32 2014 +0800
```

删掉了c.txt文件

```
commit fed1d562614e581bcb4b8bb925408f01e039e113
Author: longen0707 <879083421@qq.com>
Date: Mon Oct 20 14:56:07 2014 +0800
```

删掉了c.txt文件

```
commit 7fcb8ee84f9d4d5ebae7b001424794108956424a
Author: longen0707 <879083421@qq.com>
Date: Mon Oct 20 14:40:15 2014 +0800
```

添加b.txt文件

```
commit d8bb7b49d053019e4807d427756d8e1331cb2fef
Author: longen0707 <879083421@qq.com>
Date: Mon Oct 20 14:27:40 2014 +0800
```

添加文件a.txt

```
commit 4612fa5c71b1ece119c75199702add45ba5c3157
Author: longen0707 <879083421@qq.com>
Date: Mon Oct 20 11:53:26 2014 +0800
```

一次性提交所有文件，包括新建文件test.txt

```
commit 6fcfc898c63c2c760ea75865312f6242baa2ac92
Author: longen0707 <879083421@qq.com>
Date: Mon Oct 20 10:56:35 2014 +0800
```

添加readme.txt文件内容为333333