第八天git

一、回顾知识点

1.1 互斥和同步概述

同步和互斥是用于解决如下两个问题：

1）在多任务操作系统中，同时运行的多个任务可能都需要访问/使用同一种资源。

2）多个任务之间有依赖关系，某个任务的运行依赖于另一个任务。

互斥:一个公共资源同一时刻只能被一个进程或线程使用，多个进程或线程不能同时使用公共资源。

同步:两个或两个以上的进程或线程在运行过程中协同步调，按预定的先后次序运行。

【注意】同步是特殊的互斥。

1.2 互斥锁

> 用于线程的互斥。
>
> `#include <pthread.h>`

mutex 互斥锁是一种简单的加锁的方法来控制对共享资源的访问。

互斥锁只有两种状态,即加锁(lock)和解锁（unlock）。

`mutex` 用 `pthread_mutex_t` 数据类型表示，在使用互斥锁前,必须先对它进行初始化。

```
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER
int pthread_mutex_init(pthread_mutex_t *mutex,
                       const pthread_mutexattr_t  *attr);

int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_trylock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t * mutex);
int pthread_mutex_destroy(pthread_mutex_t *mutex);
```

死锁：

```
1） 上锁之后，未解锁
2） 多把锁的上锁顺序错误，如某一个线程上锁1，等待获取锁2；而另一个线程上锁2，等待锁1.
```

1.3 读写锁

> 如果就两个任务，一个读、一个写，建议使用互斥锁。
>
> 如果3个或3个以上的任务，多个读或多个写，建议使用读写锁。

读写锁的基本特点如下：

- 多个线程可以同时获取读取锁，实现并发的读取操作。
- 写入锁是独占的，当有线程获取写入锁时，其他线程无法获取读取锁或写入锁。
- 当有线程持有读取锁时，其他线程可以继续获取读取锁，但不能获取写入锁。

- 读取锁和写入锁之间的优先级关系可以根据具体的实现策略而定，如读优先或写优先。

> 读写锁的类型： `pthread_rwlock_t`

```
#include <pthread.h>
int pthread_rwlock_init(pthread_rwlock_t *restrict_rwlock,
                        const pthread_rwlockattr_t *restrict_attr);
pthread_rwlock_t my_rwlock = PTHREAD_RWLOCK_INITIALIZER;
int pthread_rwlock_destroy(pthread_rwlock_t *rwlock);

int pthread_rwlock_rdlock(pthread_rwlock_t *rwlock);
int pthread_rwlock_tryrdlock(pthread_rwlock_t *rwlock);
int pthread_rwlock_wrlock(pthread_rwlock_t *rwlock);
int pthread_rwlock_trywrlock(pthread_rwlock_t *rwlock);
int pthread_rwlock_unlock(pthread_rwlock_t *rwlock);
```

1.4 条件变量

> 条件变量是用来等待而不是用来上锁的，条件变量本身不是锁。通常条件变量和互斥锁同时使用。
>
> 条件变量的两个动作： 条件不满足, 阻塞线程; 当条件满足, 通知阻塞的线程开始工作。
>
> 条件变量的类型: `pthread_cond_t`。

```
#include <pthread.h>
int pthread_cond_init(pthread_cond_t *restrict cond,
                      const pthread_condattr_t *restrict attr);
pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
int pthread_cond_destroy(pthread_cond_t *cond);

int pthread_cond_wait(pthread_cond_t *restrict_cond,
                      pthread_mutex_t *restrict_mutex);
int pthread_cond_timedwait(pthread_cond_t *restrict cond,
                           pthread_mutex_t *restrict mutex,
                           const struct timespec *restrict abstime);

int pthread_cond_signal(pthread_cond_t *cond);
int pthread_cond_broadcast(pthread_cond_t *cond);
```

1.5 信号量

信号量广泛用于进程或线程间的同步和互斥，信号量本质上是一个非负的整数计数器，它被用来控制对公共资源的访问。

当信号量值大于 0 时，则可以访问，否则将阻塞。`PV` 原语是对信号量的操作，一次 P 操作使信号量减１，一次 V 操作使信号量加１。

信号量数据类型为：`sem_t`

信号量完成互斥

> 不管有多少个任务，只要是互斥，只要一个信号量，并且初始化1.

信号量用于同步

> 有几个任务就需要有几个信号量，先执行为任务的信号初始化为1，其他信号量初始化为0，所有任务P 自己的信号，V 下一个任务的信号量。

```
#include <semaphore.h>
int sem_init(sem_t *sem, int pshared, unsigned int value)

int sem_wait(sem_t *sem);   // P操作
int sem_trywait(sem_t *sem);

int sem_post(sem_t *sem); // V操作

int sem_destroy(sem_t *sem);
int sem_getvalue(sem_t *sem, int *sval);
```

1.6 有名信号量

POSIX 的信号量有两种：

- 1、无名信号量 (即信号量，用于线程间同步或互斥)
- 2、有名信号量(用于进程间同步或互斥)

```
#include <fcntl.h>
#include <sys/stat.h>
#include <semaphore.h>

//打开有名信号量
sem_t *sem_open(const char *name, int oflag);
sem_t *sem_open(const char *name, int oflag,
                       mode_t mode, unsigned int value);

int sem_close(sem_t *sem);  // 关闭信号量
int sem_unlink(const char *name); // 删除信号量文件
```

二、git的应用

2.1 git概念

> git 是一个代码协同管理工具, 用于多人协同开发时管理代码。
>
> git 是一个开源分布式版本控制系统，可用于高效的管理大小项目，由林纳斯（Linux 之父）于2005年创造发明的。
>
> 常见的版本管理工具：svn、git。

作用：

- 防止代码丢失，可以做备份
- 代码版本的管理，可以进行多个版本之间的跳跃
- 可以方便的将代码在多人之间进行共享传输
- 多人开发时，有各种模式可以方便代码管理

分布式和集中式

- 分布式
  每一个结点都保存完整的代码，没有统一的中央服务器，节点之间相互推送下载代码完成代码共

享，例如 git

- 集中式
  代码集中管理，每次完成的代码上传到中央管理器，然后再统一从中央管理器中下载代码使用，例如 svn

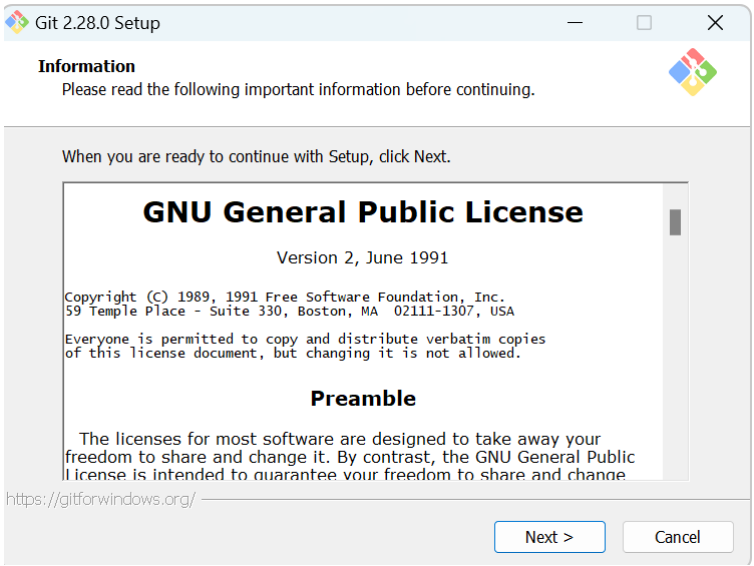
2.2 git 的特点

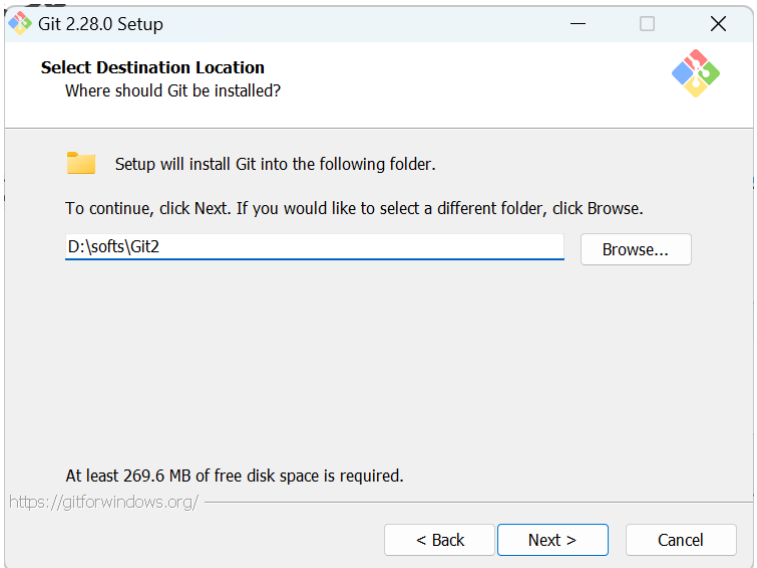
- git 可以管理各种文件，特别是代码项目，多在 Linux 和 Unix 系统中使用是分布式管理，不同于集中式，这是 git 和 svn 的核心区别
- git 可以很好的支持分支，方便多人协作工作
- git 分布式代码分家安全，有全球唯一的 commit 版本号
- git 是开源的系统
- 使用 git 可以脱网工作，且数据传输速度较快

2.3 git安装

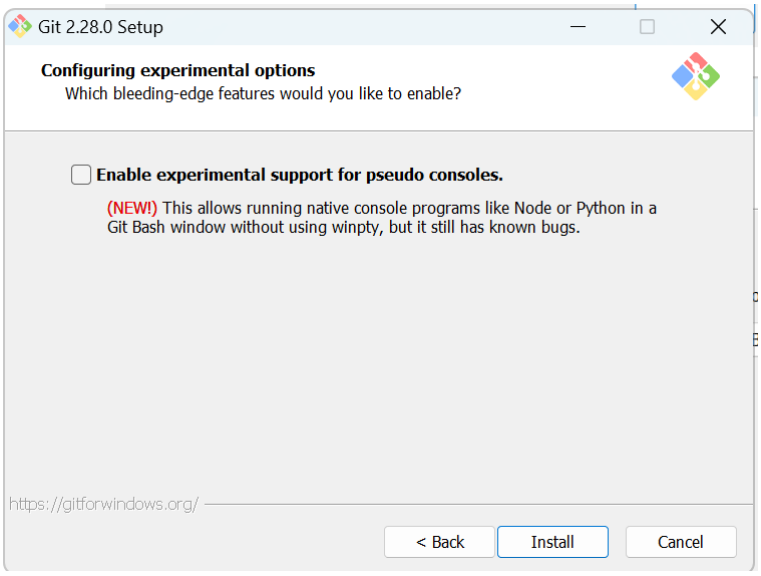
2.3.1 Window下安装

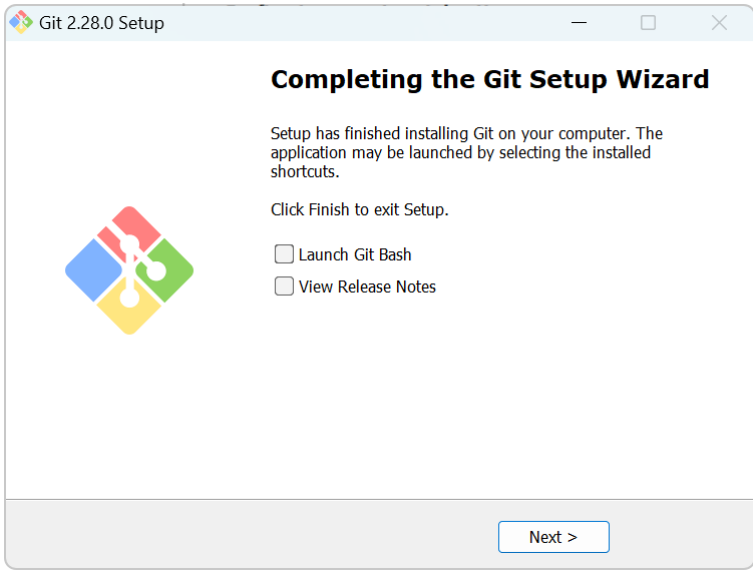
双击`Git-2.28.0-64-bit.exe`，打开安装向导界面，不遇到安装路径，一般都点击【下一步】。



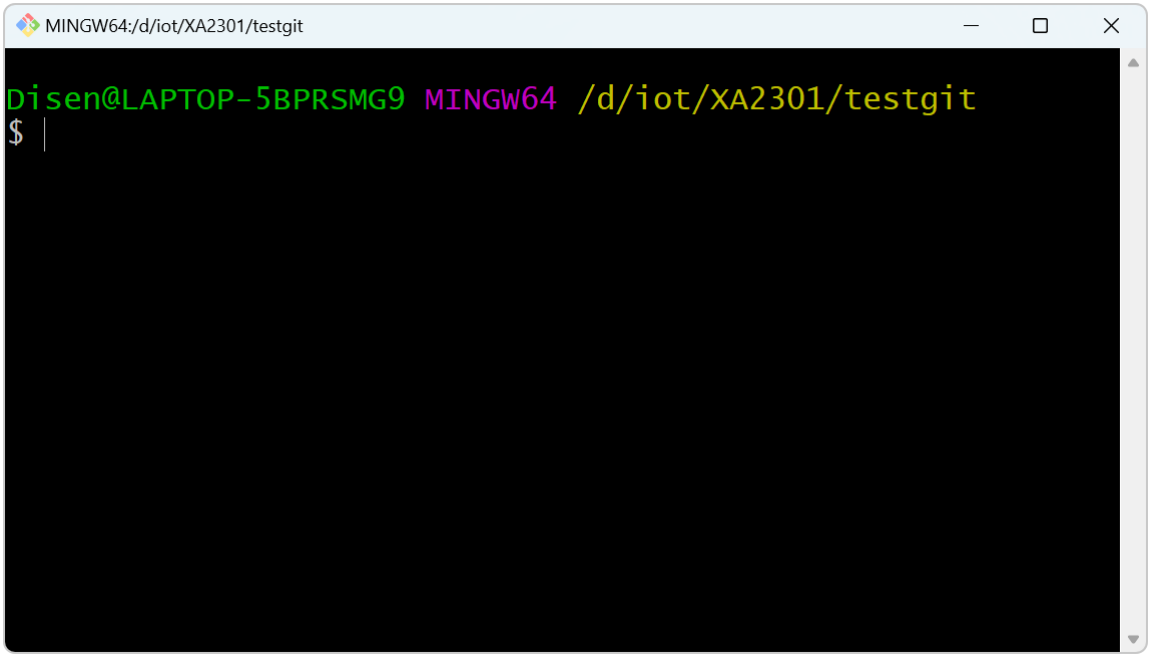


一路【Next】之后，到安装界面，点击【Install】即开始安装。

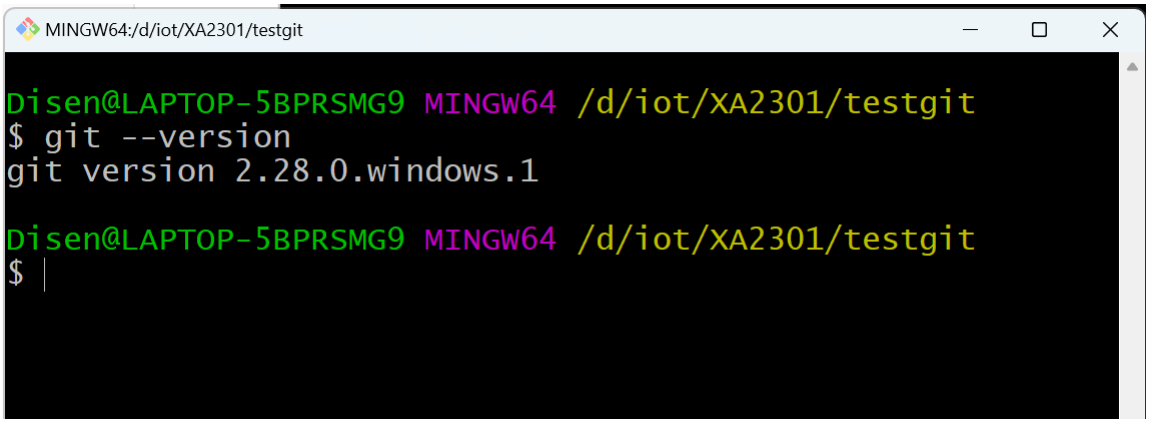




在Window，先创建一个文件夹，进入到这个文件夹中，右键空白处， 选择`git bash here`



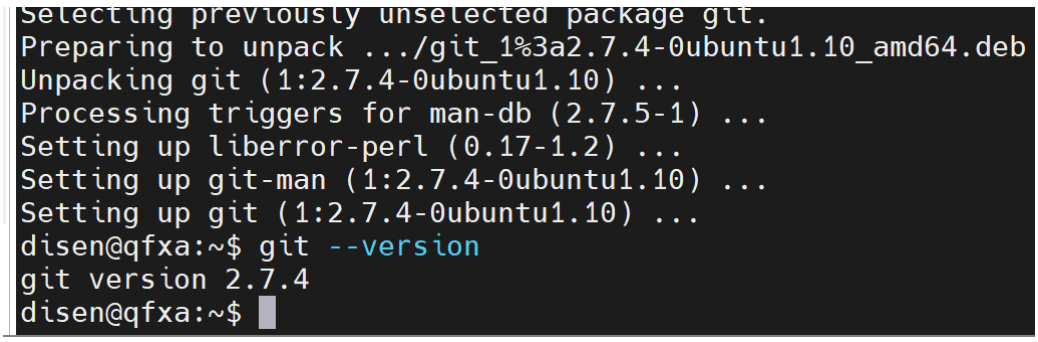
可以在窗口，执行git的相关命令：



2.3.2 Linux下安装

在ubuntu下安装git

```
sudo apt install git -y
```



查看git的版本

```
git --version
```

2.4 git配置

> 主要配置git本地仓库的用户名和邮箱
>
> git config

三种配置级别：

```
git config --system  系统级别（所有用户可共享的配置）
    Linux:  /etc/gitconfig
git config --global  用户级别(当前终端所属用户的) 【建议】
    Linux:  ~/gitconfig
git config           项目级别(当前所在的文件夹或目录)
    Window/Linux:  ./.git/config
```

2.4.1 查看所有配置信息

```
git config [--system|global]  -l
```

```
Disen@LAPTOP-5BPRSMG9 MINGW64 /d/iot/XA2301/testgit
$ git config -l
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
http.sslbackend=openssl
http.sslcainfo=D:/softs/Git2/mingw64/ssl/certs/ca-bundle.crt
core.autocrlf=true
core.fscache=true
core.symlinks=false
pull.rebase=false
credential.helper=manager
```

2.4.2 配置用户名和邮箱

```
git config --global user.name "disenQF"
git config --global user.email "610039018@qq.com"
```

```
Disen@LAPTOP-5BPRSMG9 MINGW64 /d/iot/XA2301/testgit
$ git config --global user.name "disenQF"

Disen@LAPTOP-5BPRSMG9 MINGW64 /d/iot/XA2301/testgit
$ git config --global user.email "610039018@qq.com"

Disen@LAPTOP-5BPRSMG9 MINGW64 /d/iot/XA2301/testgit
$ git config -l
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
http.sslbackend=openssl
http.sslcainfo=D:/softs/Git2/mingw64/ssl/certs/ca-bundle.crt
core.autocrlf=true
core.fscache=true
core.symlinks=false
pull.rebase=false
credential.helper=manager
user.name=disenQF
user.email=610039018@qq.com

Disen@LAPTOP-5BPRSMG9 MINGW64 /d/iot/XA2301/testgit
```

```
disen@qfxa:~/testgit$ git config --global user.name "disenQF"
disen@qfxa:~/testgit$ git config --global user.email "610039018@qq.com"
disen@qfxa:~/testgit$ git config -l
user.name=disenQF
user.email=610039018@qq.com
disen@qfxa:~/testgit$
```

2.5 git的基本操作

2.5.1 初始化git

```
git init
```

在某个目录下初始化仓库后会自动产生.git 目录，改目录下工作的所有文档可使用 git 进行管理。【实际上创建本地的仓库， 标识当前目录下存在一个.git目录】。

【注意】如果当前已存在.git目录，则先删除再初始化。

```
disen@qfxa:~/testgit$ git init
Initialized empty Git repository in /home/disen/testgit/.git/
disen@qfxa:~/testgit$ ls -a
.  ..  .git
disen@qfxa:~/testgit$
```

2.5.2 查看仓库状态

可以查看当前仓库的当前分支（默认主分支）上的未提交的资源状态， 命令的结果中包含建议执行的命令。

```
git status
```

```
Disen@LAPTOP-5BPRSMG9 MINGW64 /d/iot/XA2301/testgit (master)
$ git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)

Disen@LAPTOP-5BPRSMG9 MINGW64 /d/iot/XA2301/testgit (master)
```

2.5.3 添加文件

可以将当前目录中所有的文件或指定的文件添加到暂存区

```
git add 文件名|.|*
```

```
disen@qfxa:~/testgit$ touch a.txt
disen@qfxa:~/testgit$ git status
On branch master

Initial commit

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

        a.txt

nothing added to commit but untracked files present (use "git add" to track)
disen@qfxa:~/testgit$ git add a.txt
disen@qfxa:~/testgit$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   a.txt

disen@qfxa:~/testgit$
```

2.5.4 删除暂存区的文件

在没有提交之前，可以将已添加到暂存区的文件删除

```
git rm --cached 文件名|-r .|*
```

```
disen@qfxa:~/testgit$ git rm --cached *
rm 'a.txt'
disen@qfxa:~/testgit$ git add .
disen@qfxa:~/testgit$ git rm --cached a.txt
rm 'a.txt'
disen@qfxa:~/testgit$ git status
On branch master

Initial commit

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

        a.txt

nothing added to commit but untracked files present (use "git add" to track)
```

2.5.5 提交暂存区

将暂存区的文件提交到本地仓库中，每一次提交都必须注释相关信息。

```
git commit -m "本次提交的信息"
```

```
disen@qfxa:~/testgit$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   a.txt

disen@qfxa:~/testgit$ git commit -m "add a.txt"
[master (root-commit) ecd01e3] add a.txt
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 a.txt
disen@qfxa:~/testgit$
```

2.5.6 查看提交版本信息

查看当前提交的版本日志

```
git log  [--pretty=oneline|email|full|fuller]  [-n 数值]
```

```
disen@qfxa:~/testgit$ git log --pretty=oneline
ecd01e38ecc6ae29c54ce342d5e3b7df48188f06 add a.txt
disen@qfxa:~/testgit$ git log --pretty=email
From ecd01e38ecc6ae29c54ce342d5e3b7df48188f06 Mon Sep 17 00:00:00 2001
From: disenQF <610039018@qq.com>
Date: Wed, 23 Aug 2023 11:10:04 +0800
Subject: [PATCH] add a.txt

disen@qfxa:~/testgit$ git log --pretty=full
commit ecd01e38ecc6ae29c54ce342d5e3b7df48188f06
Author: disenQF <610039018@qq.com>
Commit: disenQF <610039018@qq.com>

    add a.txt
disen@qfxa:~/testgit$ git log --pretty=fuller
commit ecd01e38ecc6ae29c54ce342d5e3b7df48188f06
Author:     disenQF <610039018@qq.com>
AuthorDate: Wed Aug 23 11:10:04 2023 +0800
Commit:     disenQF <610039018@qq.com>
CommitDate: Wed Aug 23 11:10:04 2023 +0800

    add a.txt
disen@qfxa:~/testgit$
```

2.5.7 比较文件

本地文件与本地的仓库文件进行比较, 显示出不同的地方

```
git diff <文件名>
```

```
disen@qfxa:~/testgit$ git diff a.txt
diff --git a/a.txt b/a.txt
index e69de29..8733ede 100644
--- a/a.txt
+++ b/a.txt
@@ -0,0 +1,2 @@
+hello,disen
+lucy, go,go!
```

```
disen@qfxa:~/testgit$ 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:   a.txt

no changes added to commit (use "git add" and/or "git commit -a")
disen@qfxa:~/testgit$ git add .
disen@qfxa:~/testgit$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   a.txt

disen@qfxa:~/testgit$ git commit -m "a.txt新增了两行"
[master 5990567] a.txt新增了两行
 1 file changed, 2 insertions(+)
disen@qfxa:~/testgit$
```

```
disen@qfxa:~/testgit$ vi a.txt
disen@qfxa:~/testgit$ git diff a.txt
diff --git a/a.txt b/a.txt
index 8733ede..a1684e4 100644
--- a/a.txt
+++ b/a.txt
@@ -1,2 +1,2 @@
-hello,disen
 lucy, go,go!
+disen, 6666!
disen@qfxa:~/testgit$
```

红色表本地仓库中的文件即将删除的部分， 绿色代表即将提交的新增部分。

2.6 git的恢复操作

2.6.1 从本地仓库恢复本地文件

如果本地文件发生改变之后，想恢复到之前提交的版本内容，可以进行恢复

```
git checkout <文件名> ...
```

```
disen@qfxa:~/testgit$ git diff a.txt
diff --git a/a.txt b/a.txt
index a1684e4..1faa87b 100644
--- a/a.txt
+++ b/a.txt
@@ -1,2 +1,2 @@
-lucy, go,go!
+jack, go,go!
 disen, 6666!
disen@qfxa:~/testgit$ git checkout a.txt
disen@qfxa:~/testgit$ git diff a.txt
disen@qfxa:~/testgit$ cat a.txt
lucy, go,go!
disen, 6666!
disen@qfxa:~/testgit$
```

2.6.2 丢弃工作区的修改

放弃当前工作目录下某一个文件的修改， 实际上同一个命令。

```
git checkout -- <文件名> ...
```

```
disen@qfxa:~/testgit$ git diff a.txt
diff --git a/a.txt b/a.txt
index a1684e4..4e96288 100644
--- a/a.txt
+++ b/a.txt
@@ -1,2 +1,3 @@
 lucy, go,go!
 disen, 6666!
+jack, 888!
disen@qfxa:~/testgit$ git checkout -- a.txt
disen@qfxa:~/testgit$ git diff a.txt
disen@qfxa:~/testgit$ cat a.txt
lucy, go,go!
disen, 6666!
disen@qfxa:~/testgit$
```

2.6.3 本地仓库文件的移动和删除

> 【注意】移动和删除后，需要再次执行 git commit

在本地仓库中将源文件移动目的目录中（本地也会移动）

```
git mv <源文件> <目的目录>
```

```
disen@qfxa:~/testgit$ git mv b.txt m1
disen@qfxa:~/testgit$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        renamed:    b.txt -> m1/b.txt

disen@qfxa:~/testgit$ git commit -m "move b.txt to m1"
[master dff92b1] move b.txt to m1
 1 file changed, 0 insertions(+), 0 deletions(-)
 rename b.txt => m1/b.txt (100%)
disen@qfxa:~/testgit$ ls
a.txt  c.txt  m1
disen@qfxa:~/testgit$
```

在本地仓库中删除指定的文件（本地也会删除）

```
git rm <文件名>
```

```
disen@qfxa:~/testgit$ git rm c.txt
rm 'c.txt'
disen@qfxa:~/testgit$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        deleted:    c.txt

disen@qfxa:~/testgit$ git commit -m "delete c.txt"
[master 415118e] delete c.txt
 1 file changed, 0 insertions(+), 0 deletions(-)
 delete mode 100644 c.txt
disen@qfxa:~/testgit$ ls
a.txt  m1
```

2.6.4 回到某一个版本

回到上一个版本(commit)

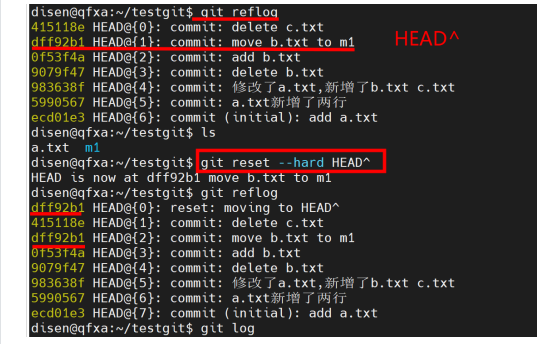
```
git reset --hard HEAD^
```

回到指定的版本(commit)

```
git reset --hard <commit id>
```

查看历史版本记录

```
git reflog
```




```
disen@qfxa:~/testgit$ git log --pretty=oneline
dff92b1c4ce614081edf5a4ac7bc840a21483378 move b.txt to m1
0f53f4ab69d5cd32bd3adc19c9b7505a0b492eff add b.txt
9079f4706f35357d1916cf6f5171b667dac3d624 delete b.txt
983638f88f448b2a013ac45ba616b4c1fec2a6e4 修改了a.txt,新增了b.txt c.txt
5990567e57799931e88e9df7cbc51db3ea5c7413 a.txt新增了两行
ecd01e38ecc6ae29c54ce342d5e3b7df48188f06 add a.txt
disen@qfxa:~/testgit$ ls
a.txt  c.txt  m1
disen@qfxa:~/testgit$ git reset --hard 9079
HEAD is now at 9079f47 delete b.txt
disen@qfxa:~/testgit$
```

表示当前仓库中唯一的版本号，至少4位

```
disen@qfxa:~/testgit$ ls
a.txt  c.txt
```

【注意】回到了某一个版本，本地文件也会发生变化，但不是真的删除，还可以通过`git reset --hard <commit id>`命令再恢复之前的版本。

2.7 标签管理

> 标签：在当前工作的位置添加快照，保存工作状态，一般用于版本的迭代

2.7.1 创建标签

```
git tag <标签名称>                    创建标签
git tag <标签名称> -m <提示信息>      创建标签并设置提示信息
git tag <标签名称> <commit id>        指定某一个 commit id 处创建标签
```

```
disen@qfxa:~/testgit$ git tag
disen@qfxa:~/testgit$ git tag "del c.txt"
fatal: 'del c.txt' is not a valid tag name.
disen@qfxa:~/testgit$ git tag tag1
disen@qfxa:~/testgit$ git tag
tag1
disen@qfxa:~/testgit$ git tag tag2 -m "delete b.txt" 9079
disen@qfxa:~/testgit$ git tag
tag1
tag2
```

标签名不能使用特殊符号

【注意】一个提交版本可以存在多个标签。

2.7.2 查看标签

```
git tag
```

2.7.3 查看详细标签

```
git show <标签名>
```

```
disen@qfxa:~/testgit$ git show tag2
tag tag2
Tagger: disenQF <610039018@qq.com>
Date:   Wed Aug 23 14:34:04 2023 +0800

delete b.txt

commit 9079f4706f35357d1916cf6f5171b667dac3d624
Author: disenQF <610039018@qq.com>
Date:   Wed Aug 23 11:53:22 2023 +0800

    delete b.txt
```

2.7.4 删除标签

对已存在的标签可以删除

```
git tag -d <标签名>
```

```
disen@qfxa:~/testgit$ git tag
tag1
tag2
tag3
disen@qfxa:~/testgit$ git tag -d tag3
Deleted tag 'tag3' (was 415118e)
disen@qfxa:~/testgit$ git tag
tag1
tag2
disen@qfxa:~/testgit$
```

2.7.5 恢复到指的标签位置

```
git reset --hard <标签名>
```

```
disen@qfxa:~/testgit$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        c.txt

nothing added to commit but untracked files present (use "git add" to track)
disen@qfxa:~/testgit$ git add .
disen@qfxa:~/testgit$ git commit -m "two add c.txt"
[master d9e61fd] two add c.txt
 1 file changed, 1 insertion(+)
 create mode 100644 c.txt
```

```
disen@qfxa:~/testgit$ git log --pretty=oneline
d9e61fdc2500e5e3d3e8ec72ea83bff28e127332 two add c.txt
415118e2376e5426553320fec3b076a0c5b41860 delete c.txt
dff92b1c4ce614081edf5a4ac7bc840a21483378 move b.txt to m1
0f53f4ab69d5cd32bd3adc19c9b7505a0b492eff add b.txt
9079f4706f35357d1916cf6f5171b667dac3d624 delete b.txt
983638f88f448b2a013ac45ba616b4c1fec2a6e4 修改了a.txt,新增了b.txt c.txt
5990567e57799931e88e9df7cbc51db3ea5c7413 a.txt新增了两行
ecd01e38ecc6ae29c54ce342d5e3b7df48188f06 add a.txt
disen@qfxa:~/testgit$ git reset --hard tag1
HEAD is now at 415118e delete c.txt
disen@qfxa:~/testgit$ ls
a.txt  m1
```

【注意】恢复的文件只限于本地仓库中的文件， 本地新增的文件并不会受到影响。

2.8 临时工作区管理 【了解】

> 用于对当前工作区（暂存区）修改但未提交的文件进行保存，以便在之后的操作时，可以恢复。
> 保存之后可以进行切换分支、恢复之前版本等操作，操作之后再恢复未完成的工作。

2.8.1 创建保存临时工作区

```
git stash [save <message>]
```

当本地仓库 中的文件修改后，并未提交时，可以保存临时工作区。

```
disen@qfxa:~/testgit$ echo "disen" >> d.txt
disen@qfxa:~/testgit$ 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:   d.txt
```

```
disen@qfxa:~/testgit$ git stash
Saved working directory and index state WIP on master: f9ecafe add d.txt
HEAD is now at f9ecafe add d.txt
```

```
disen@qfxa:~/testgit$ git status
On branch master
nothing to commit, working directory clean
```

保存临时工作区成功之后，当前的工作区（暂存区）与本地仓库保持一致。表时，修改且未提交的信息则会被保存起来。

2.8.2 查看临时工作区

```
git stash list
```

```
disen@qfxa:~/testgit$ git stash list
stash@{0}: WIP on master: f9ecafe add d.txt
disen@qfxa:~/testgit$ echo "jack" >> d.txt
disen@qfxa:~/testgit$ 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:   d.txt

no changes added to commit (use "git add" and/or "git commit -a")
disen@qfxa:~/testgit$ git stash save "jack appedn d.txt"
Saved working directory and index state On master: jack appedn d.txt
HEAD is now at f9ecafe add d.txt
disen@qfxa:~/testgit$ git stash list
stash@{0}: On master: jack appedn d.txt
stash@{1}: WIP on master: f9ecafe add d.txt
disen@qfxa:~/testgit$
```

2.8.3 使用指定的工作区

> 将之前未提交的修改的内容进行恢复，继续再修改。

```
git stash apply stash@{编号}
```

```
disen@qfxa:~/testgit$ git stash apply stash@{1}
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:   d.txt

no changes added to commit (use "git add" and/or "git commit -a")
disen@qfxa:~/testgit$ cat d.txt
abc
disen
```

2.8.4 使用最新的临时工作区并删除

```
git stash pop
```

```
disen@qfxa:~/testgit$ echo "lucy" >> d.txt
disen@qfxa:~/testgit$ 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:   d.txt

no changes added to commit (use "git add" and/or "git commit -a")
disen@qfxa:~/testgit$ git stash save "append lucy to d.txt"
Saved working directory and index state On master: append lucy to d.txt
HEAD is now at dbfc684 merge d.txt
disen@qfxa:~/testgit$ git stash pop
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:   d.txt

no changes added to commit (use "git add" and/or "git commit -a")
Dropped refs/stash@{0} (fd2b441cb2a1607dac1746d7e651612942d9833b)
```

2.8.5 删除指定工作区

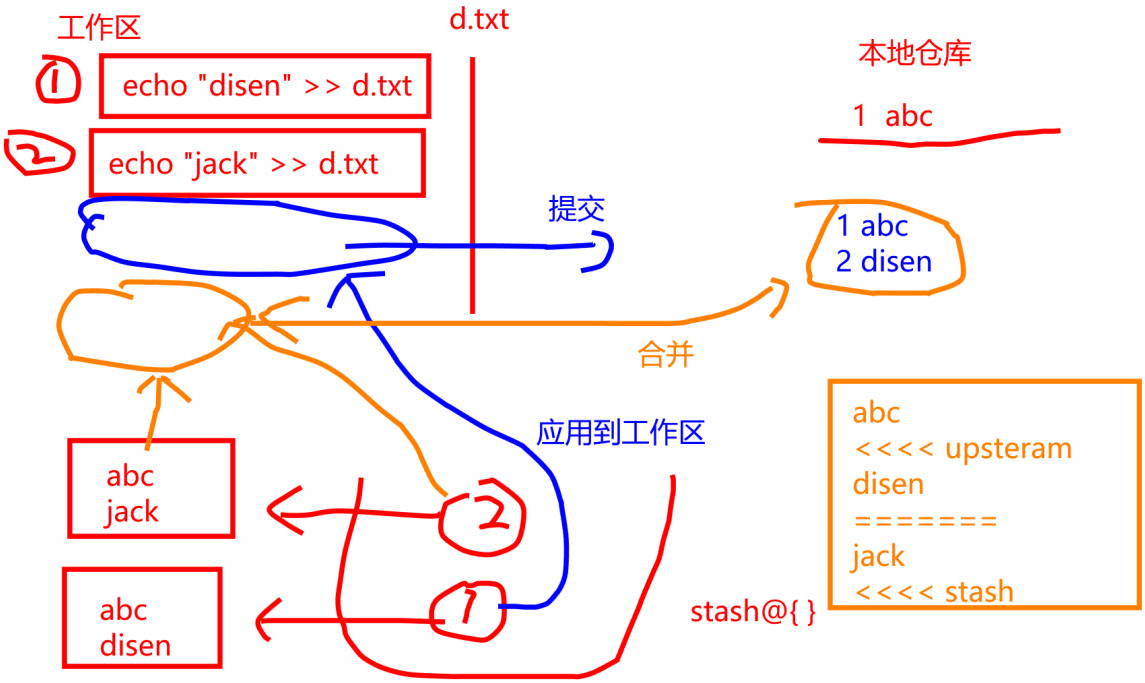
```
git stash drop stash@{编号}
```

2.8.6 删除所有的临时工作区

```
git stash clear
```

```
disen@qfxa:~/testgit$ git stash list
stash@{0}: On master: jack appedn d.txt
stash@{1}: WIP on master: f9ecafe add d.txt
disen@qfxa:~/testgit$ git stash clear
disen@qfxa:~/testgit$ git stash list
disen@qfxa:~/testgit$
```

2.8.7 画图分析git stash工作过程



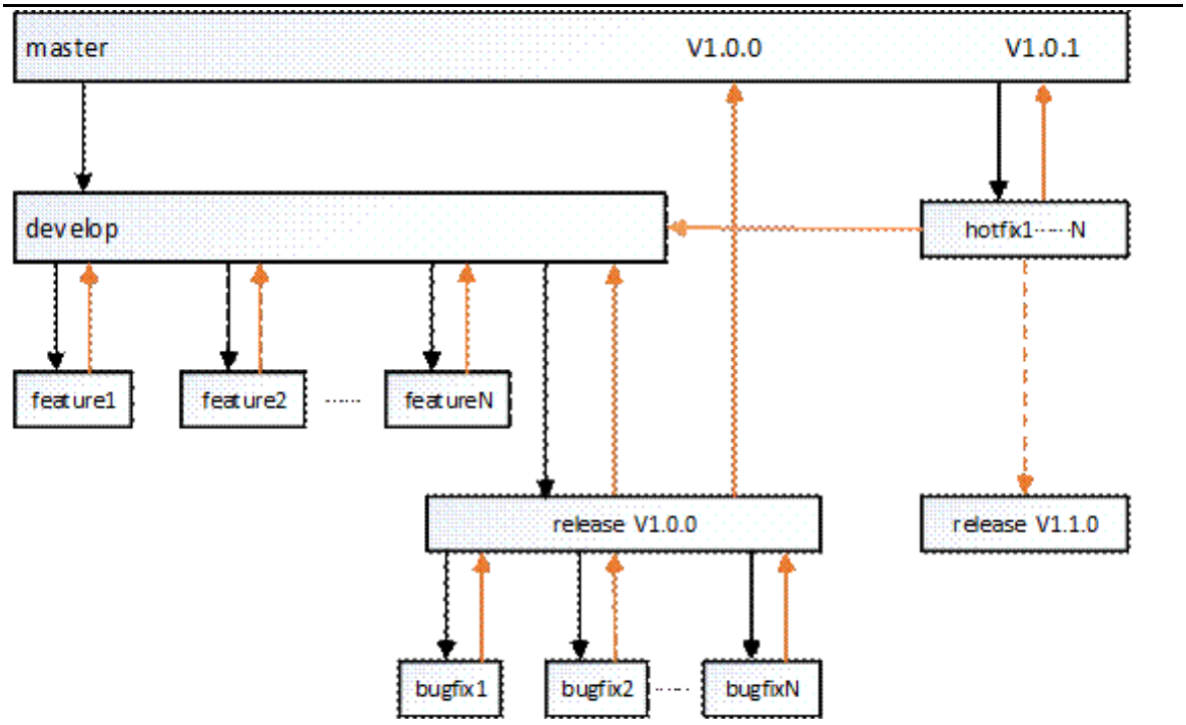
2.9 分支操作 【重点】

分支主要用于多人协作工作，分支即获取原有代码，在此基础上创建自己的工作环境，单独开发，不会影响其他分支的操作，开发完成后再统一合并到主线分支中。

创建分支的好处：安全，不影响其他人工作。

2.9.1 分支的分类

project（项目库）
- master(主线分支)
  - hotfix（线上紧急 bug 修复分支）
- develop（开发分支）
  - feature（功能分支）
  - release（发布分支）
    - bugfix（bug 修复分支）



注意：一个项目库只能有一个 master 分支和一个 develop 分支项目库创建后，由版本管理员从 master 分支创建 develop 分支，项目组所有成员需要克隆服务器上的项目库到本地，然后再从 develop 分支创建自己的 feature 分支进行工作。

2.9.2 分支操作

2.9.2.1 创建分支

在当前的分支下创建新的分支

```
git branch <分支名称>
```

2.9.2.2 查看分支

```
git branch
```

2.9.2.3 切换分支

```
git checkout <分支名>
```

2.9.2.4 创建并切换分支

```
git checkout -b <分支名>
```

```
disen@qfxa:~/testgit$ git branch dev
disen@qfxa:~/testgit$ git branch
  dev
* master
disen@qfxa:~/testgit$ git checkout dev
Switched to branch 'dev'
disen@qfxa:~/testgit$ git branch
* dev
  master
disen@qfxa:~/testgit$ git log -n 1
commit 274a2f88ec601b802d7bf7c977a5555881988f95
Author: disenQF <610039018@qq.com>
Date:   Wed Aug 23 16:37:09 2023 +0800

    modify d.txt
disen@qfxa:~/testgit$ ls
a.txt  d.txt  m1
```

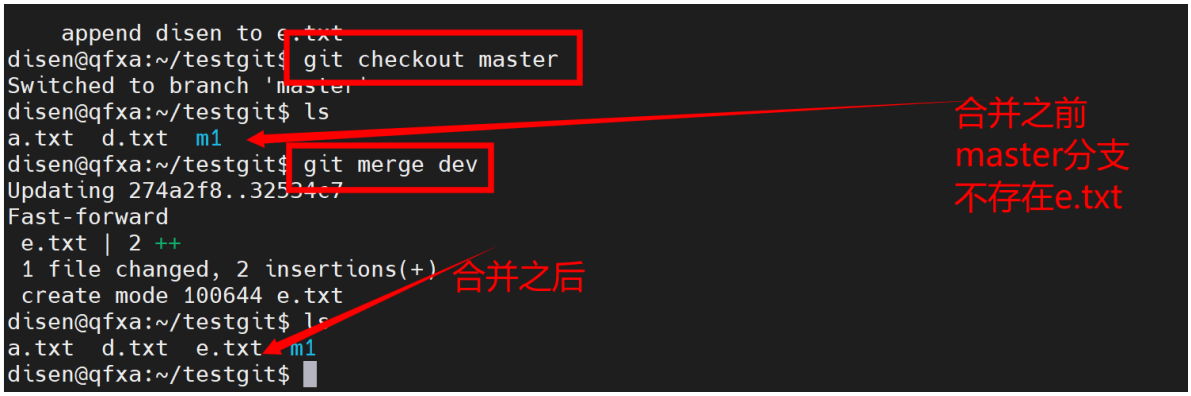
```
disen@qfxa:~/testgit$ echo "good" > e.txt
disen@qfxa:~/testgit$ git add .
disen@qfxa:~/testgit$ git commit -m "add e.txt"
[dev 0bcada5] add e.txt
 1 file changed, 1 insertion(+)
 create mode 100644 e.txt
disen@qfxa:~/testgit$ git log -n 1
commit 0bcada5708c54bcf2984b480d38ed2b5102b5d87
Author: disenQF <610039018@qq.com>
Date:   Wed Aug 23 16:41:09 2023 +0800

    add e.txt
disen@qfxa:~/testgit$ ls
a.txt  d.txt  e.txt  m1
disen@qfxa:~/testgit$ cat e.txt
good
disen@qfxa:~/testgit$
```

2.9.2.5 合并分支

> 将指定的分支合并到当前分支上

```
git merge <分支名>
```



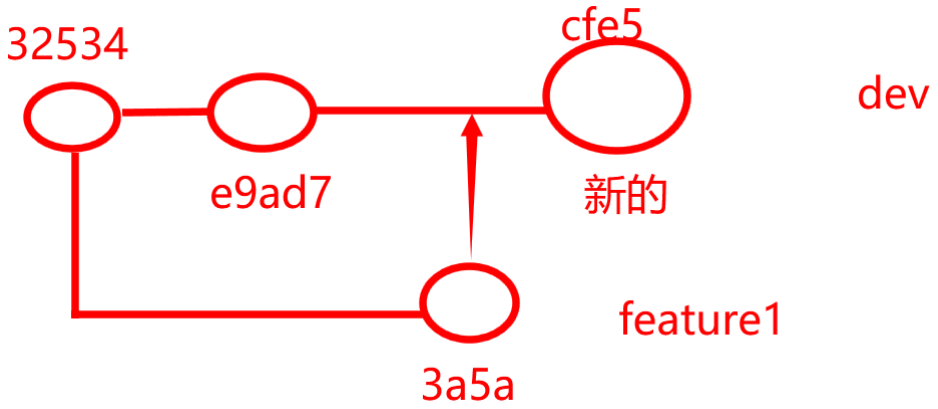
2.9.2.6 衍合分支

> 将指定的分支衍合到当前分支上

```
git rebase <分支名>
```

合并操作:

```
dev分支下(32534...):   git branch feature1
dev分支下(32534...):   echo "hi" > f.txt
dev分支下(32534...):   git add .
dev分支下(32534...):   git commit -m "add f.txt"
dev分支下(e9ad7...):   git checkout feature1
feature1分支下(32534...):   touch h.txt
feature1分支下(32534...):   git add . && git commit -m "add h.txt"
feature1分支下(3a5a...):   git checkout dev
dev分支下(e9ad7...):       git merge feature1
dev分支下(cfe5...):
```



友好查看分支情况：

```
git log --color --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s
%Cgreen(%cr) %C(bold blue)<%an>%Creset' --abbrev-commit
```

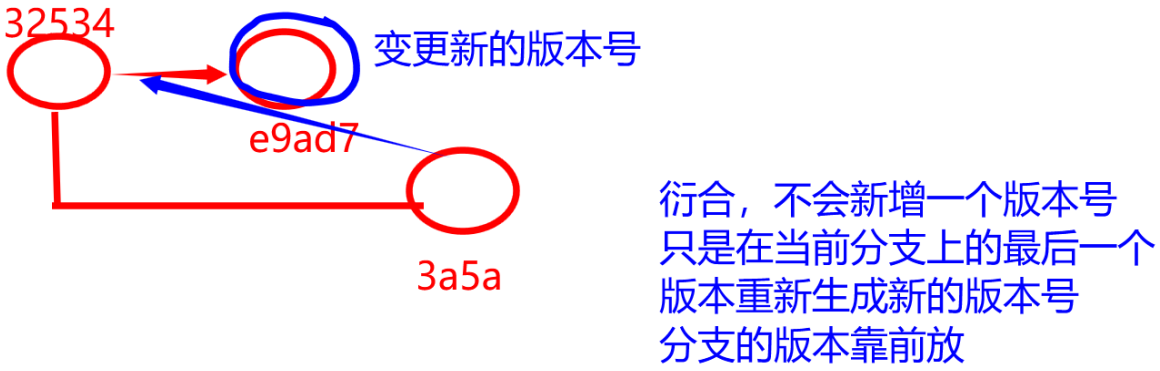
```
disen@qfxa:~/testgit$ git log --color --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %
Cgreen(%cr) %C(bold blue)<%an>%Creset' --abbrev-commit
*   cfe5001 - (HEAD -> dev) Merge branch 'feature1' into dev (8 minutes ago) <disenQF>
|\
| * 3a5a400 - (feature1) add h.txt (9 minutes ago) <disenQF>
* | e9ad742 - add f.txt (17 minutes ago) <disenQF>
|/
* 32534c7 - (master) append disen to e.txt (26 minutes ago) <disenQF>
* 0bcada5 - add e.txt (29 minutes ago) <disenQF>
* 274a2f8 - modify d.txt (33 minutes ago) <disenQF>
* dbfc684 - merge d.txt (86 minutes ago) <disenQF>
* 44edd65 - append disen to d.txt (2 hours ago) <disenQF>
* f9ecafe - add d.txt (2 hours ago) <disenQF>
* 415118e - (tag: tag1) delete c.txt (3 hours ago) <disenQF>
* dff92b1 - move b.txt to m1 (3 hours ago) <disenQF>
* 0f53f4a - add b.txt (3 hours ago) <disenQF>
* 9079f47 - (tag: tag2) delete b.txt (5 hours ago) <disenQF>
* 983638f - 修改了a.txt,新增了b.txt c.txt (5 hours ago) <disenQF>
* 5990567 - a.txt新增了两行 (6 hours ago) <disenQF>
* ecd01e3 - add a.txt (6 hours ago) <disenQF>
```

扩展： 在`~/.gitconfig`中配置友好显示分支情况

```
[alias]
lg = log --color --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s
%Cgreen(%cr) %C(bold blue)<%an>%Creset' --abbrev-commit
```

衍合操作：

```
dev分支下(32534...):   git branch feature1
dev分支下(32534...):   echo "hi" > f.txt
dev分支下(32534...):   git add .
dev分支下(32534...):   git commit -m "add f.txt"
dev分支下(e9ad7...):    git checkout feature1
feature1分支下(32534...):    touch h.txt
feature1分支下(32534...):    git add . && git commit -m "add h.txt"
feature1分支下(3a5a...):   git checkout dev
dev分支下(e9ad7...):       git rebase feature1
dev分支下(ee0ed...):
```



2.9.2.7 删除分支

```
git branch -d|D < 分支名称>
```

如果删除的分支没有合并，是不能用当前命令删除的, 使用-D 删除，表示强制删除指定分支。

```
disen@qfxa:~/testgit$ git branch -d feature1
Deleted branch feature1 (was 3a5a400).
disen@qfxa:~/testgit$ git branch
* dev
  master
```

2.10 远程仓库

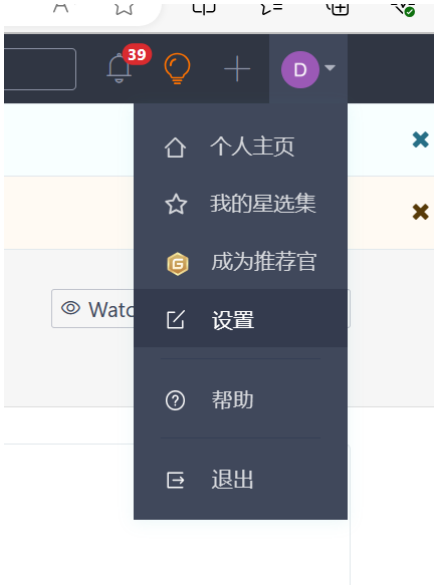
> 需要远程仓库的服务器，建议使用gitee服务。

2.10.1 gitee仓库创建

打开浏览器，输入`gitee.com`,先登录，登录成功之后

本地生成公钥文件（ssh-keygen命令生成的），将公钥文件的内容复制到gitee上。

```
disen@qfxa:~/testgit$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/disen/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/disen/.ssh/id_rsa.
Your public key has been saved in /home/disen/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:IBe+pcKwJMnTJwC8Jt2Q0J5kWpOoku0ehhrfEwJiO2M disen@qfxa
The key's randomart image is:
+---[RSA 2048]----+
|== o  .          |
|oo%  . .         |
|+%+B..+ .        |
|O**++o =         |
|*+o o o S        |
|oE= ..           |
|o=o+ .           |
|. o o            |
|     .           |
+----[SHA256]-----+
disen@qfxa:~/testgit$
```



2.10.2 远程仓库的操作

2.10.2.1 设置本地仓库的远程仓库位置

```
git remote add  <仓库名称>  远程仓库的地址
```

如:

```
git remote add origin git@gitee.com:disenQF/testgitiot2301.git
```

```
disen@qfxa:~/testgit$ git remote add origin git@gitee.com:disenQF/testgitiot2301.git
disen@qfxa:~/testgit$ git config -l
user.name=disenQF
user.email=610039018@qq.com
alias.lg=log --color --graph --pretty=format:'%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %
ld blue)<%an>%Creset' --abbrev-commit
core.repositoryformatversion=0
core.filemode=true
core.bare=false
core.logallrefupdates=true
remote.origin.url=git@gitee.com:disenQF/testgitiot2301.git
remote.origin.fetch=+refs/heads/*:refs/remotes/origin/*
disen@qfxa:~/testgit$
```

2.10.2.2 上传本地仓库的文件

第一次本地仓库上传到远程仓库时的命令：

```
git push -u origin master
```

```
disen@qfxa:~/testgit$ git push -u origin master
Counting objects: 37, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (24/24), done.
Writing objects: 100% (37/37), 3.29 KiB | 0 bytes/s, done.
Total 37 (delta 1), reused 0 (delta 0)
remote: Powered by GITEE.COM [GNK-6.4]
To git@gitee.com:disenQF/testgitiot2301.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.
disen@qfxa:~/testgit$
```

之后再上传到远程仓库的命令：

```
git push
```

强制覆盖远程仓库的文件（强制上传）：

```
git push -u --force origin  master
```

```
disen@qfxa:~$ cd code3
disen@qfxa:~/code3$ git init
Initialized empty Git repository in /home/disen/code3/.git/
disen@qfxa:~/code3$ git add .
disen@qfxa:~/code3$ git commit -m "init code3"
[master (root-commit) edcaed7] init code3
 130 files changed, 3371 insertions(+)
```

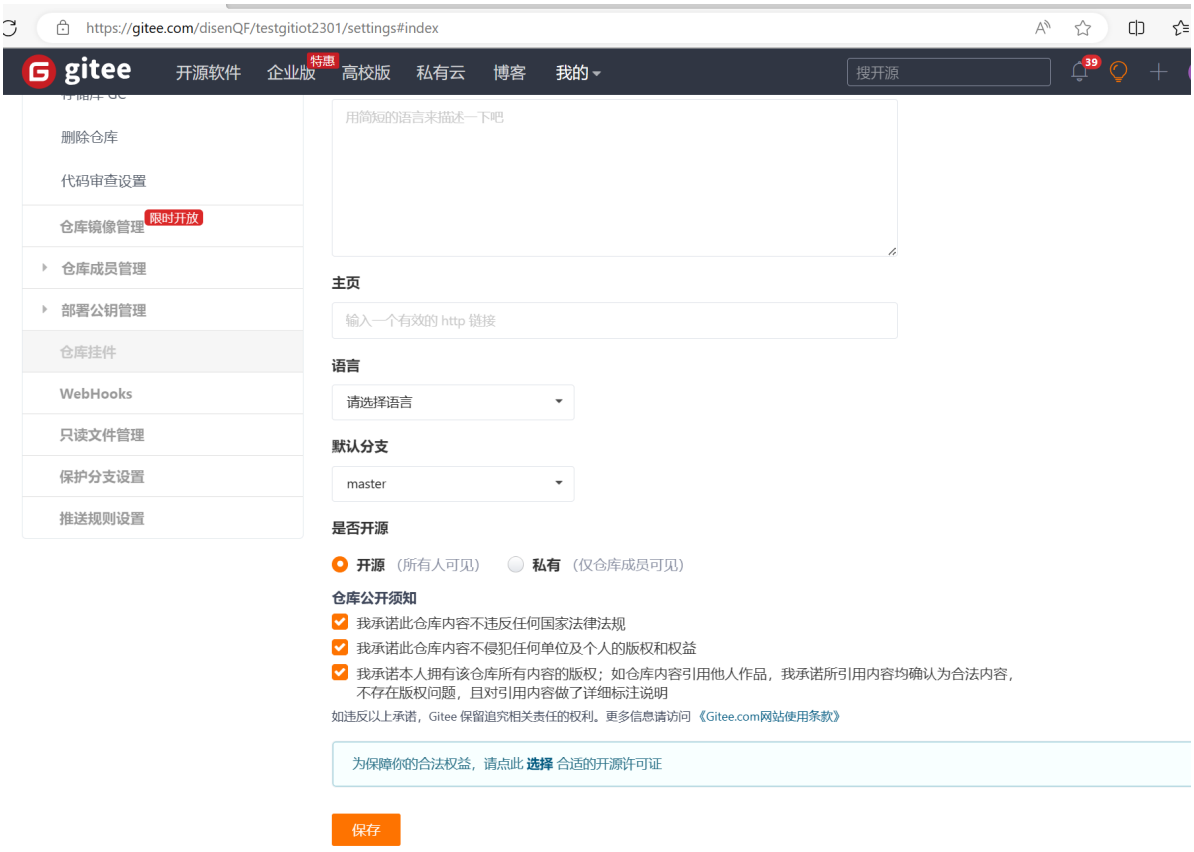
```
disen@qfxa:~/code3$ git remote add origin git@gitee.com:disenQF/testgitiot2301.git
disen@qfxa:~/code3$ git push -u origin master
To git@gitee.com:disenQF/testgitiot2301.git
 ! [rejected]        master -> master (fetch first)
error: failed to push some refs to 'git@gitee.com:disenQF/testgitiot2301.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
disen@qfxa:~/code3$ git push -u --force origin master
Counting objects: 139, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (135/135), done.
Writing objects: 100% (139/139), 121.02 KiB | 0 bytes/s, done.
Total 139 (delta 48), reused 0 (delta 0)
remote: Resolving deltas: 100% (48/48), done.
remote: Powered by GITEE.COM [GNK-6.4]
To git@gitee.com:disenQF/testgitiot2301.git
 + 32534c7...edcaed7 master -> master (forced update)
Branch master set up to track remote branch master from origin.
disen@qfxa:~/code3$
```

2.10.2.3 远程仓库下载

第一次下载:

```
git clone 远程仓库的地址(http, ssh)
```

如果远程仓库给其他人下载，必须先设置它为开源的




```
disen@qfxa:~$ git clone https://gitee.com/disenQF/testgitiot2301.git
Cloning into 'testgitiot2301'...
remote: Enumerating objects: 139, done.
remote: Counting objects: 100% (139/139), done.
remote: Compressing objects: 100% (87/87), done.
remote: Total 139 (delta 48), reused 139 (delta 48), pack-reused 0
Receiving objects: 100% (139/139), 121.02 KiB | 0 bytes/s, done.
Resolving deltas: 100% (48/48), done.
Checking connectivity... done.
```

其次下载远程仓库的最新的文件：

```
git pull
```

修改远程仓库的地址：