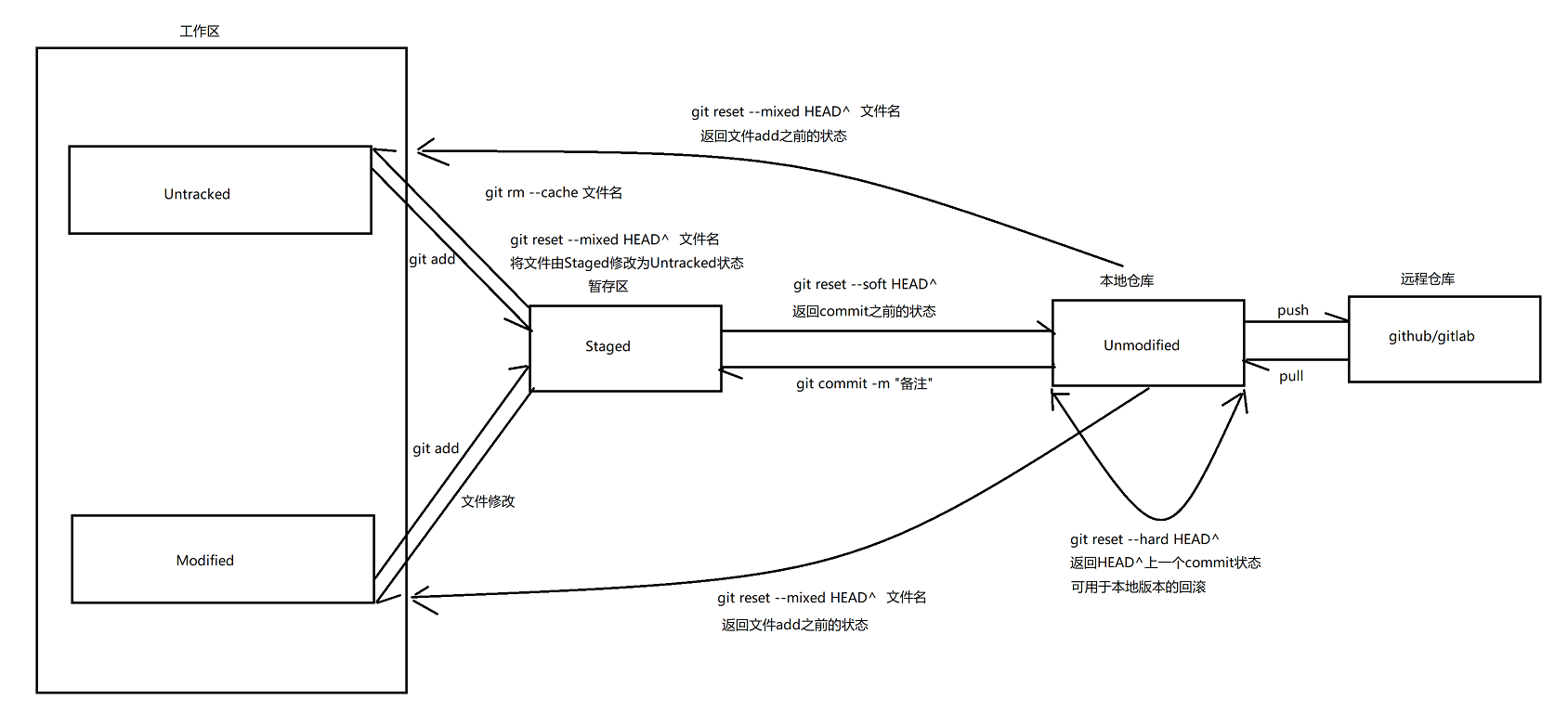
# 1.git概述

## 1.1区域概述



## 1.2使用案例

#查看git版本

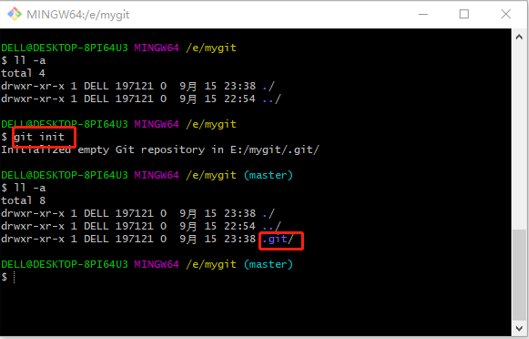
git –version

#创建本地仓库

mkdir TestGit

cd TestGit

git init



#新建文本README.md

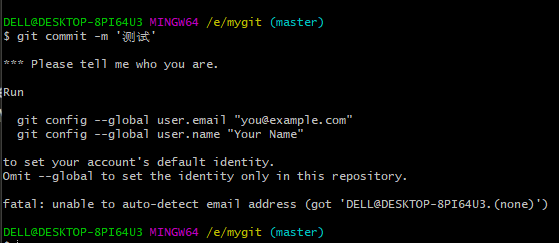
git "该仓库用于测试Git脚本" > README.md

#将新建的文本放入暂存区

git add README.md

#将暂存区文件放入本地仓库

git commit -m '测试'



#初次使用在向仓库提交的时候需要指明email(邮箱)和name(作者名)

git config --global user.email "sunjunjun@example.com"

git config --global user.name "helloworld"

#设置好email和name后，重新提交

git commit -m '测试'

#拉取远程仓库(初次拉取GitHub远程仓库需要输入GitHub的账户名和密码)

git remote add origin <https://github.com/Bear-AI-Hand/TestGit.git>

#将本地仓库往名字为origin的仓库的master分支上提交变更

git push -u origin master

#拉取远程仓库的变更到本地仓库，查看变化修改情况

git fetch

#将远程的变更，合并到本地仓库的master分支

git merge origin/master

#不建议使用 等同于fetch之后merge

git pull

小结：

新建文件，文件处于Untracked ，Untracked文件不被Git跟踪

修改本地仓库文件后，文件处于Modified

git add命令将Untracked和Modified修改为Staged 状态，加入到暂存区

git commit命令将Staged 文件(暂存区)提交到本地仓库，文件处于Unmodified

git rm --cache 将文件修改为Untracked状态，不被Git跟踪

git pull 拉取远程仓库代码

git push 上传本地仓库代码到远程仓库

# 2.git命令

## 2.1基本命令

#查看git文件状态

git status

# 新建文件，文件处于Untracked

echo 'helloworld' > 1.txt

#添加文件到暂存区，Untracked -> Staged，后期commit的时候会上传到本地仓库

git add 1.txt

#删除文件到暂存区，Untracked -> Staged，后期commit的时候会从本地仓库删除

git rm 1.txt

#只要不提交，可以从本地仓库master分支上恢复数据

git checkout master 1.txt

# 提交文件到本地仓库， staged -> Unmodified

git commit -m '新建文件'

# git rm --cache，并非删除文件，目的是将文件由Unmodified和Staged 状态转换成Untracked

git rm --cache 1.txt

# 为什么会有这个命令呢？

# Git在commit的时候不允许有Unmodified文件，可允许当前文件中有Staged状态和Untracked状态。，Staged状态会被提交到本地仓库，而Untracked状态不会被提交到本地仓库，通过add命令转换成Staged状态被提交，通过git rm --cache 命令转换成Untracked状态不会被提交

## 2.2git branch

#查看分支，不加任何参数，列出所有的分支

git branch

#增加分支

git branch 分支名

#删除分支，不能删除当前所在的分支

git barnch -d 分支名

#修改分支名

git branch -m 旧分支名 新分支名

#切换分支

git checkout 分支名

如果在分支a上面对文件进行修改之后，没有commit就切换到另外一个分支b， 这个时候会报错，因为修改的文件没有commit的文件，在切换的分支的时候会被以前文件覆盖，修改并不会进行保存，所以Git 报错提示告知必须commit后才能切换分支。

git checkout -f 分支名 强制切换到分支，如果当前有为提交的变更，会直接丢弃 -f 参数一定一定要非常非常小心使用，一般情况下不建议使用，除非真的要强制去执行。

最好的方式是在切换分支的时候，将本地修改的文件进行提交。

## 2.3git log

git log --online 输出简要的信息

git log用于查看git的提交历史

git log -数字 表示查看最近几次的提交

git log -p -数字 显示最近几次提交的不同点

git log --author 查看具体某个作者的提交

## 2.4git diff

#不加任何参数，比较同一文件在工作区(Modified)跟暂存区(Staged)的差异

git diff

#用于比较暂存区和本地仓库的差异

git diff --cached –staged

#用于比较暂存区和HEAD的差异

git diff HEAD

#查看指定文件在暂存区(Staged)和本地仓库(Unmodified)差异

git diff 文件名

#查看当前分支跟指定的分支的差异

git diff 分支名

#查看两个指定分支(已提交的)的差异，分支2 跟分支1的差别

git diff 分支名1 分支名2

#用于列出两个历史提交的差异

git diff commitid1 commitid2

#用于罗列有变更的文件

git diff --stat

#案例分析

|  |
| --- |
| $ git diff  warning: LF will be replaced by CRLF in 1.txt.  The file will have its original line endings in your working directory  **diff --git a/1.txt b/1.txt**  **index b83bdb1..f2264e0 100644**  **--- a/1.txt**  **+++ b/1.txt**  @@ -2,12 +2,13 @@  2  3  4  -5  6  7  8  9  10  +15  +20  11  12  13 |

**diff --git a/1.txt b/1.txt**

**比较暂存区(Staged)a/1.txt和的工作区(Modified)b/1.txt文件异同**

**--- a/1.txt**

**+++ b/1.txt**

@@ -2,12 +2,13 @@

**表示暂存区a/1.txt的**2,12**行发生变化，变成工作区的b/1.txt的**2,13**行，其他未改变**

2

3

4

-5

6

7

8

9

10

+15

+20

11

12

13

对**暂存区的1.txt做了修改后变更成工作区的1.txt，状态由Staged -> Modified，**

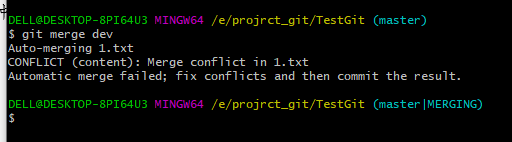
**修改操作是删除了5新增了15 20**

## 2.5git merge

#分支合并

git merge 分支名

#merge发生冲突



#解决分支冲突，用于查看产生冲突的文件

git diff --name-only --diff-filter=U

