Git

得先他妈网页上设置个XXXX.git然后才能git

1.learn route

<mark>用于版本控制</mark>

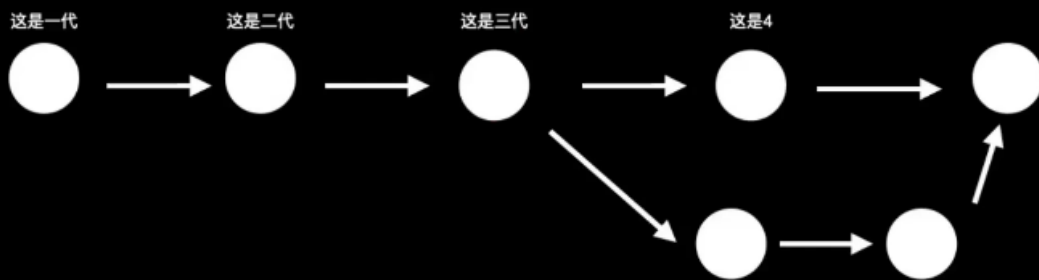用法：记住命令行多用，出错之后git项目删了，把当前编辑代码存下来，从网上down一个新的，怼里试试就能用了。

2.Git原理构成

每一个版本，一个snapshot（快照）

Git维护代码使用DAG（directed acyclic graph）有向无环图

snapshot本质上是一个commit

Thinking of history: story of snapshots

- // Skip the definition of snapshots now
- Git: directed acyclic graph (DAG)
  - simple form: a snapshot refers to a set of parents
  - Snaptshots are called "commit"s

snapshot是一堆文件和文件夹的目录

file在里称为blob

目录是个tree，可以包含blob和tree

Data model as Code

```
// a file is a bunch of bytes
type blob = array<byte>
// a directory contains named files and directories
type tree = map<string, tree | blob>
// a commit has parents, metadata, and the top-level tree

type commit = struct {
  parents: array<commit>
  author: string
  message: string
  snapshot: tree
}
```

tree里那个map<string, tree|blob>意思是用一个名字映射到tree里的blob（还是blob里的tree）

commit，一个commit可能有很多个父亲，当前的author就是当前创建这个代码的人的信息，message是附加信息，snapshot是整个项目里面的根目录

Objects and content-addressing

All types, *e.g.*, *blob*, *tree*, or *commit*, are called objects in Git

```
type object = blob | tree | commit
objects = map<string, object>
def store(object):
    id = sha1(object)
    objects[id] = object
def load(id):
    return objects[id]
```

Objects are addressed by SHA-1 hash

这仁都是git里的object

定位这个object的方法使用SHA-1这个哈希码去定位的

References as Code

```
references = map<string, string>
def update_reference(name, id):
    references[name] = id
def read_reference(name):
    return references[name]
```

```
def load_reference(name_or_id):
    if name_or_id in references:
        return load(references[name_or_id])
    else:
        return load(name_or_id)
```

人能读的SHA-1是叫references

The last piece: Repositories & Staging Area

• A Git repository: objects and references

• Why staging area?

　• Clean snapshots

　　• Git: allowing you to specify which modifications should be included in the next snapshot through a mechanism called the "staging area".

整个项目是啥

repository仓库里就是object和references

staging area选择哪些东西是要包括在下个snapshot里的空间

3.Command

用法:看介绍

Scenario场景

echo写入命令

git commit创建一个新的snapshot

查看日志

切换版本，就是当前改烂了，切换到原来版本

回到原来的版本的方法，后面加版本哈希码

git diff查看有什么修改

commit msg(message)的书写

按上面格式写

branching创建分支和merge分支
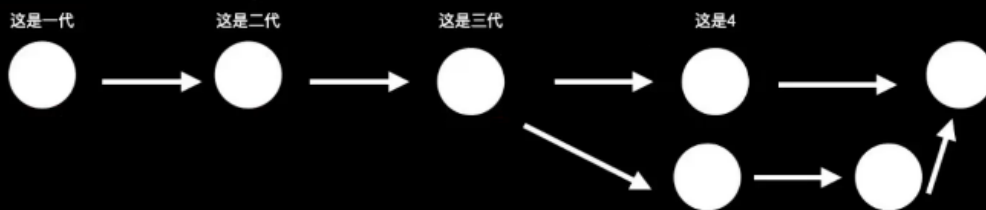
创建分支

Scenario-2: Debugging

- You find a bug in your project
- You need to add many logs to debug
- Create and switch to a new branch: *git checkout -b <name>*
- Chekc the current branch: *git branch*

```
dd@dd-PC7 ~/devlop/git-tutorial <main>
$ git status
On branch main
nothing to commit, working tree clean
dd@dd-PC7 ~/devlop/git-tutorial <main>
$ git checkout -b debug
Switched to a new branch 'debug'
dd@dd-PC7 ~/devlop/git-tutorial <debug>
$
```
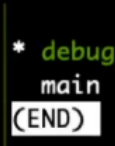
```
* debug
  main
(END)
```

git checkout -b XXX创建一个分支

git branch

<mark>合并分支merge</mark>

Scenario-2: Debugging

- Merge debug branch into main: *git merge <revision>*

```
dd@dd-PC7 ~/devlop/git-tutorial <debug>
$ git checkout main
Switched to branch 'main'
dd@dd-PC7 ~/devlop/git-tutorial <main>
$ git merge debug
Updating 78db867..86a9fe1
Fast-forward
 world.txt | 1 +
 1 file changed, 1 insertion(+)
```

只有debug有改变，main没变情况merge

<mark>！！！！！合并多个不同分支</mark>

Scenario-2: Debugging

- When you rush papers, you may have many branches, implementing *features, test cases, debug infos*

- *git rebase: Rebase is thought as one of the most complicated part in Git*

- 简单来说，*rebase*是让你在*git*维护的历史*DAG*上调整他们的结构/关系的

例如：完成这么个事

Scenario-2-1: Debugging

- Case-1: you want to keep master and topic branches, but applies commits in topic branches based on latest master commits

```
      A---B---C topic                              A'--B'--C' topic
     /                                            /
D---E---F---G master                     D---E---F---G master
```
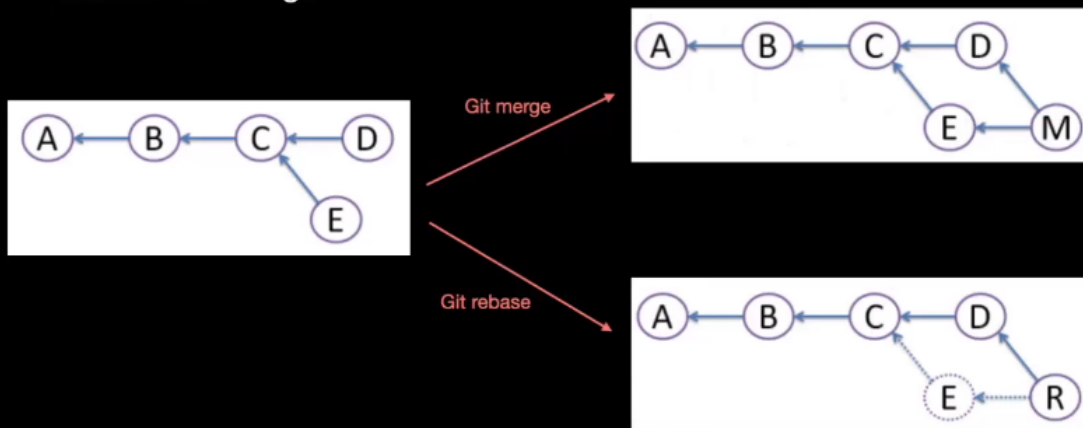
git rebase master topic

使用git rebase master topic

git merge 和 git rebase的区别（箭头反正看）

Scenario-2-1: Debugging

- Rebase vs. Merge

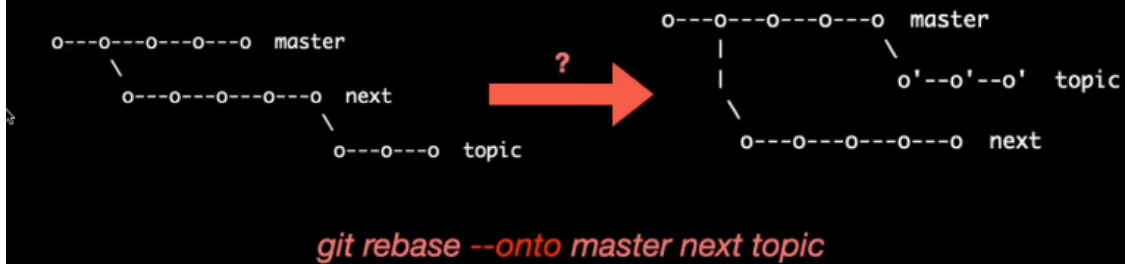在rebase里直接改的现在的commit D，而merge是自动创建个新的commit M

更复杂一个例子：

onto相当于加个reference把从next到topic的干过去
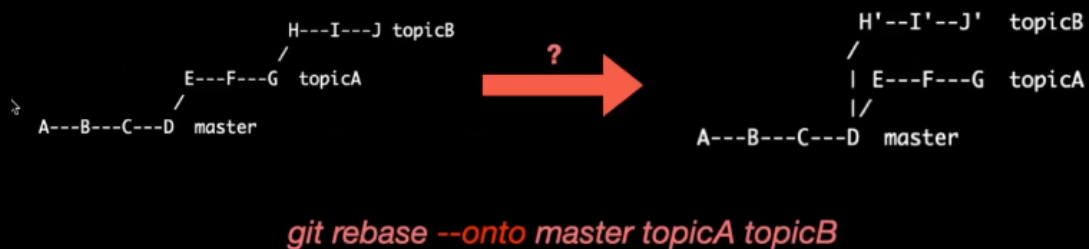

更改区间位置

骚操作，利用git rebase删掉几个commit

Command (finally...3

- Remotes

  - *git remote*: list remotes
  - *git remote add <name> <url>*: add a remote
  - *git push <remote> <local branch>:<remote branch>*: send objects to remote, and update remote reference
  - *git branch --set-upstream-to=<remote>/<remote branch>*: set up correspondence between local and remote branch
  - *git fetch*: retrieve objects/references from a remote
  - *git pull*: same as git fetch; git merge
  - *git clone*: download repository from remote

Scenario-3: Gitlab/Gitee/Github

- 定期的pull/push是个好习惯
- PR
  - 在代码仓库平台上合并修改
  - 代码Review

```
简易的命令行入门教程:

Git 全局设置:

git config --global user.name "DongDu"
git config --global user.email "dd_nirvana@sjtu.edu.cn"

创建 git 仓库:

mkdir git-tutorial
cd git-tutorial
git init
touch README.md
git add README.md
git commit -m "first commit"
git remote add origin git@gitee.com:dongduResearcher/git-tutorial.git
git push -u origin master

已有仓库?

cd existing_git_repo
git remote add origin git@gitee.com:dongduResearcher/git-tutorial.git
git push -u origin master
```

Undo撤销相关的操作

Command (finally...4

- Undo

  - *git commit --amend*: edit a commit's contents/message
  - *git reset HEAD <file>*: unstage a file
  - *git checkout -- <file>*: discard changes

修改上个commit（--amend有俩杠）

修改stage area里的文件但是保存

彻底删掉stage area里的有修改动作文件

高级操作

git blame，查某个改动是谁做的

git stash push/pop/list

干到一般需要干别的，把这个push进去然后干自己的，最后再pop出来接着干

实际上相当于后台创建了个commit

Scenario-5: Git can do more for you

- You are writing the code, but your "boss" demands that you fix something immediately : *git stash push/pop/list*

- How it works?

  - A stash entry is represented as a commit whose tree records the state of the working dir/

  - H is the HEAD commit

  - I is a commit that records the state of the index

  - W is a commit that records the state of the working tree

```
            .----W
           /    /
-----H----I
```
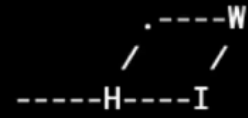
不要把二进制文件放到repo里

Scenario-5: Git can do more for you

- DO NOT UPLOAD YOU BINARY FILES TO PROJECTS!: .o, .a, .so
- *.gitignore: ignore the matched files*

```
 1 # Object files
 2 *.o
 3 *.a
 4 *.dep
 5
 6 #Build & install directories
 7 build/
 8 install/
 9
10 # Development friendly files
11 tags
```