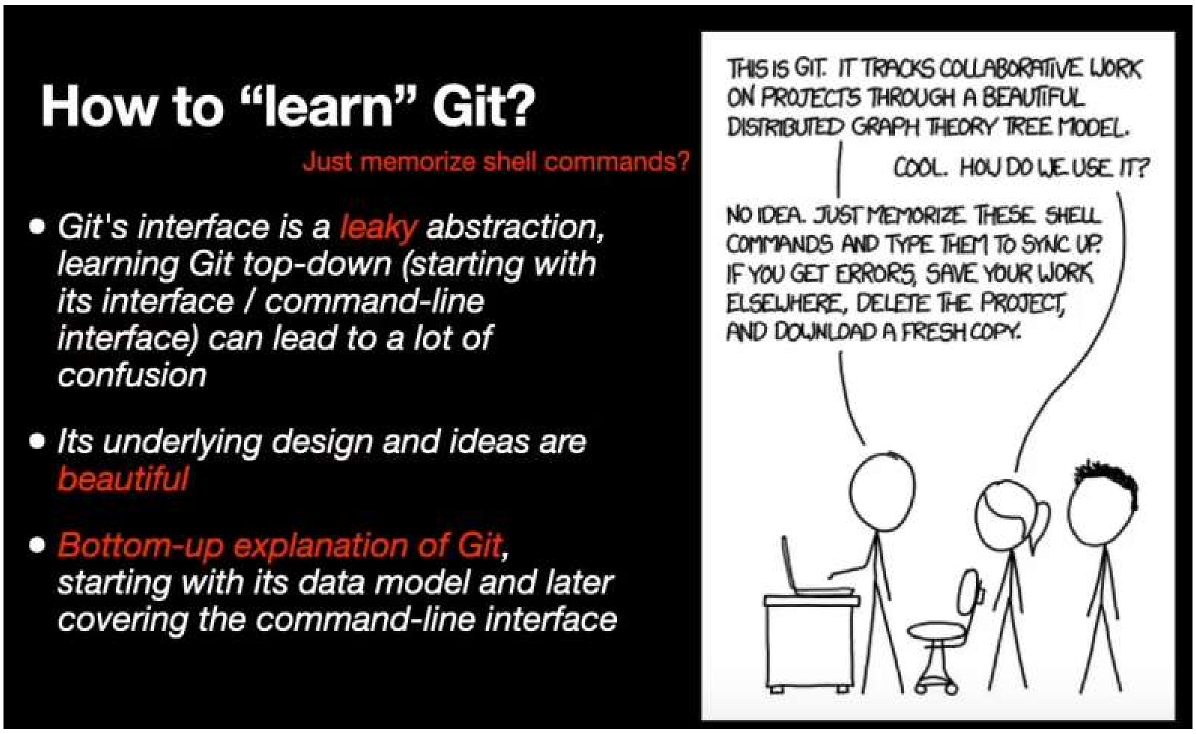
Git

得先他妈网页上设置个XXXX.git然后才能git

1.learn route



How to "learn" Git?

Just memorize shell commands?

- *Git's interface is a leaky abstraction, learning Git top-down (starting with its interface / command-line interface) can lead to a lot of confusion*
- *Its underlying design and ideas are beautiful*
- *Bottom-up explanation of Git, starting with its data model and later covering the command-line interface*

THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.

用于版本控制

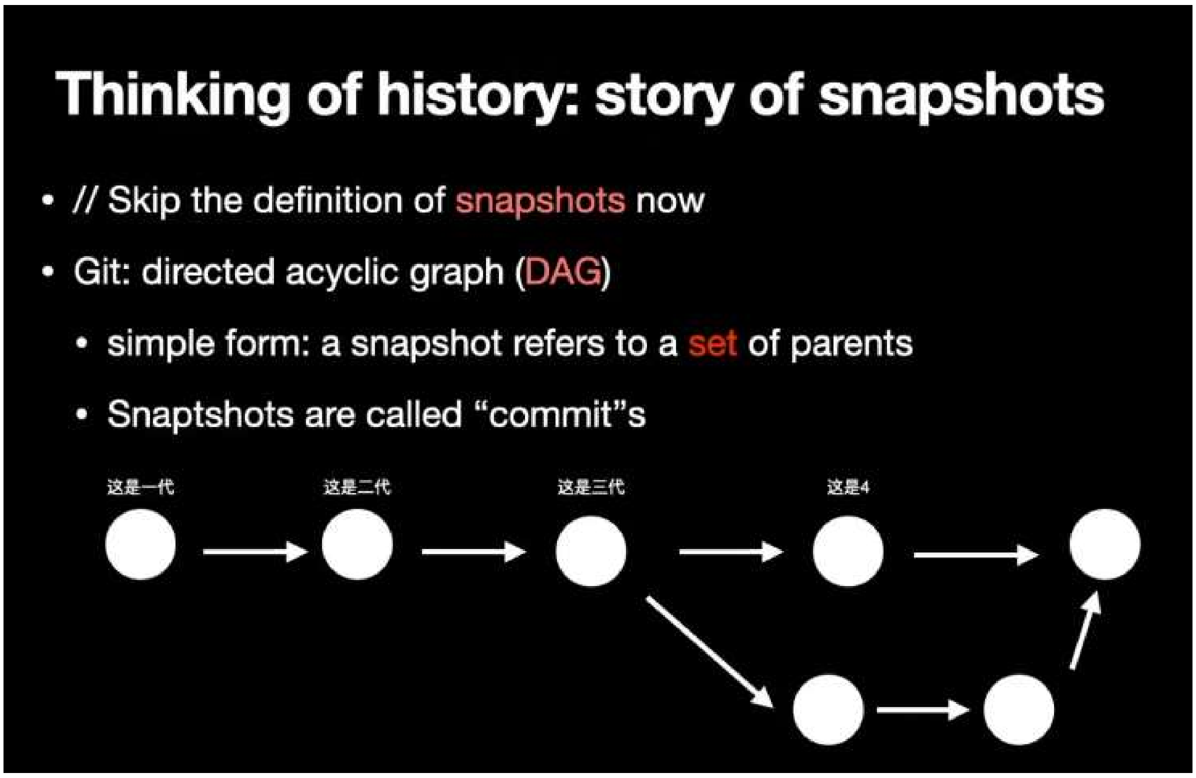
用法：记住命令行多用，出错之后git项目删了，把当前编辑代码存下来，从网上down一个新的，怼里试试就能用了。

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



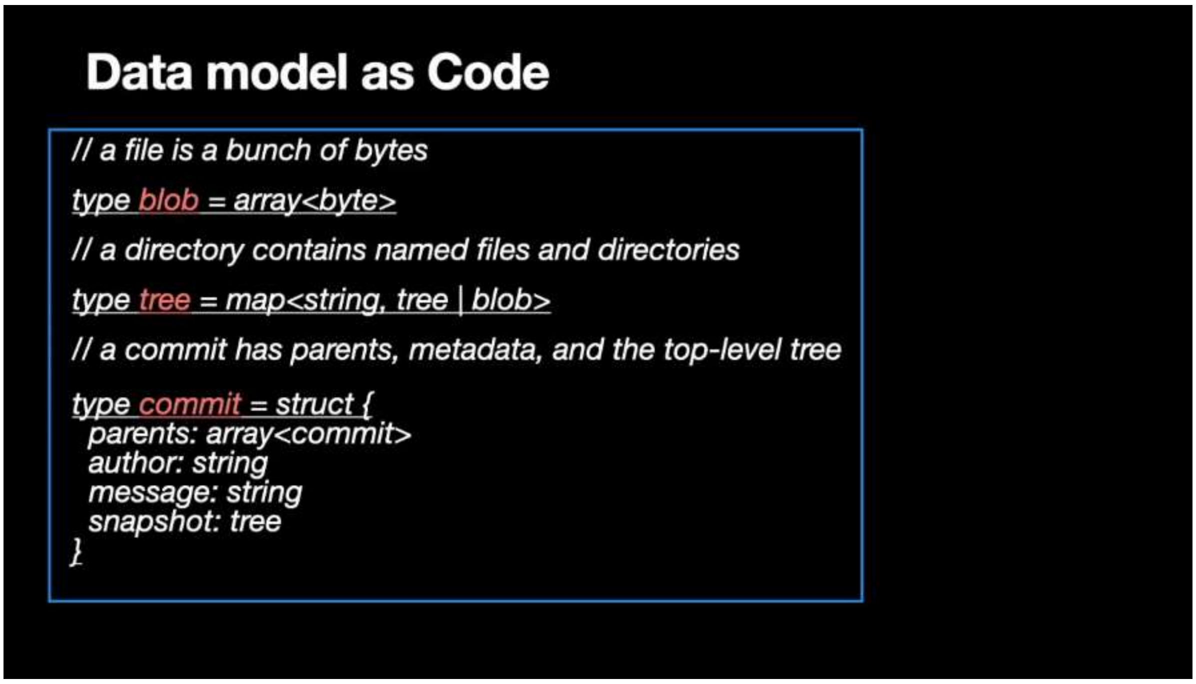
Thinking of history: story of snapshots

- // 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

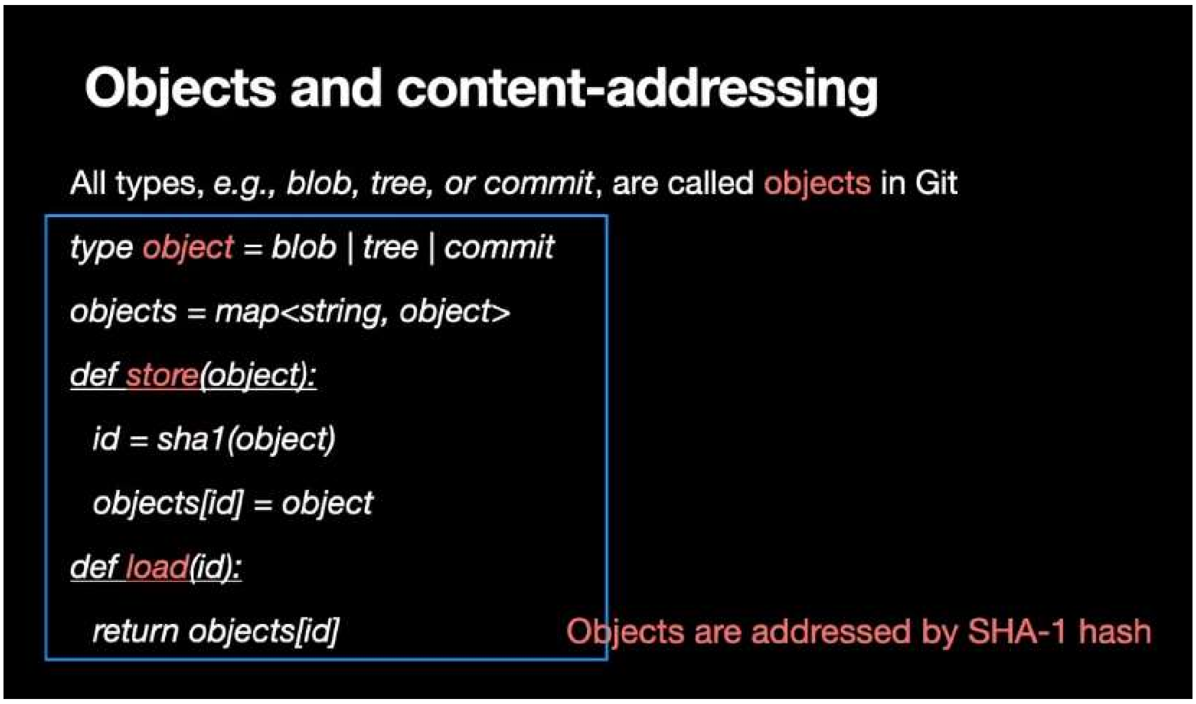


Data model as Code

```
// 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

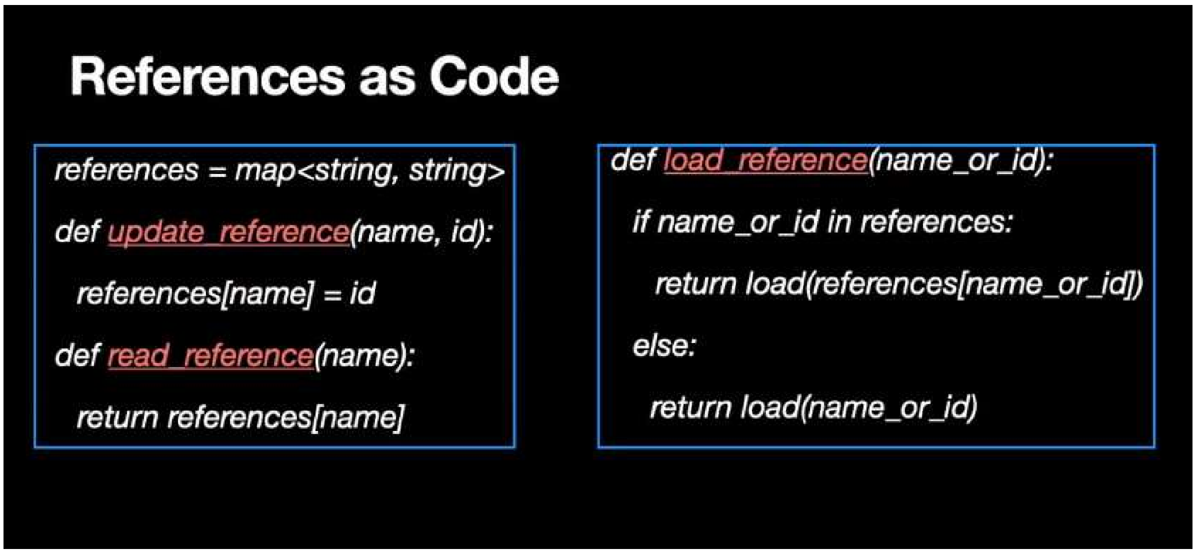
All types, *e.g., blob, tree, or commit*, are called objects in Git

```
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

这仨都是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

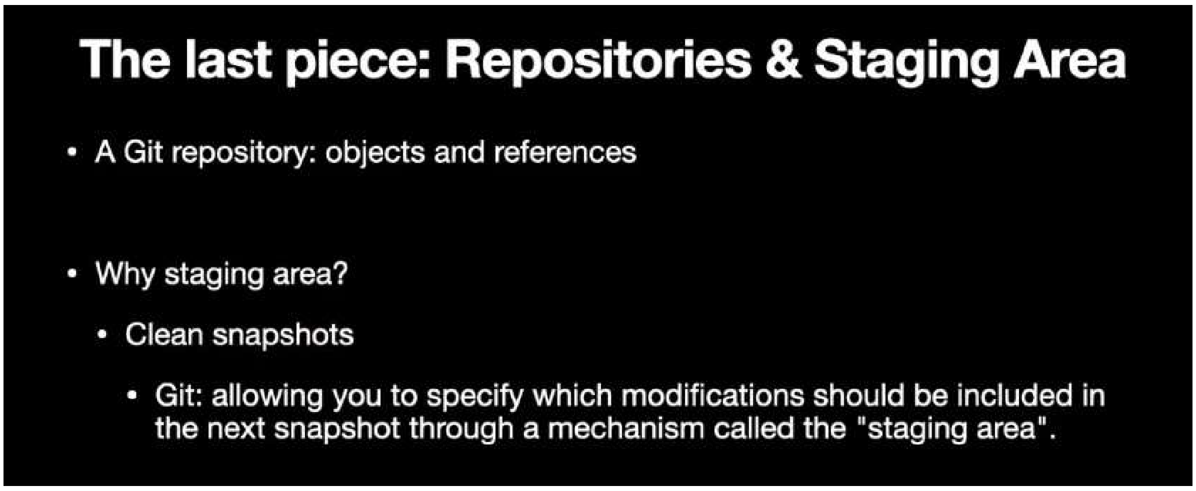


References as Code

```
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

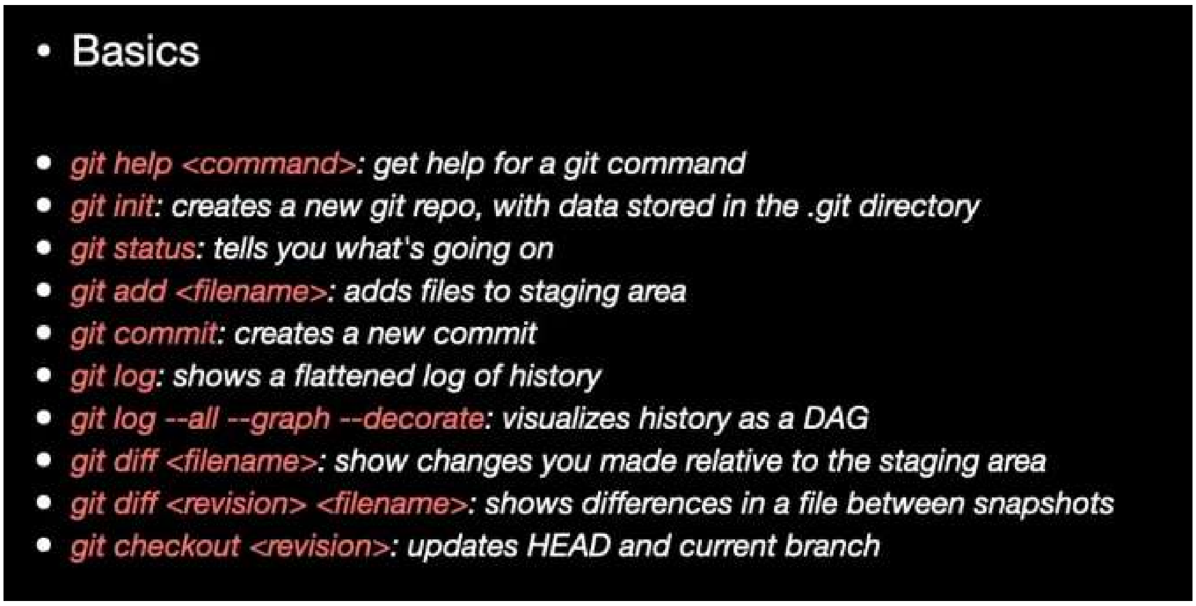
- A Git repository: objects and references
- Why staging area?
  - Clean snapshots
    - Git: allowing you to specify which modifications should be included in the next snapshot through a mechanism called the "staging area".

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

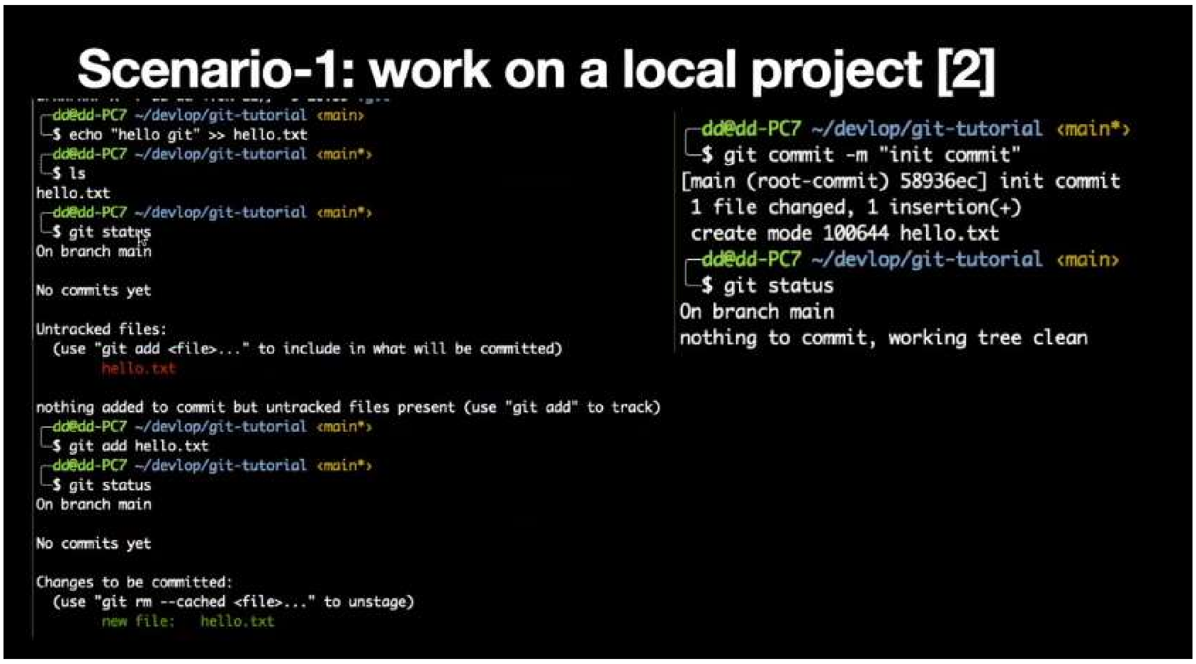
3.Command



- Basics

- git help <command>: get help for a git command
- git init: creates a new git repo, with data stored in the .git directory
- git status: tells you what's going on
- git add <filename>: adds files to staging area
- git commit: creates a new commit
- git log: shows a flattened log of history
- git log --all --graph --decorate: visualizes history as a DAG
- git diff <filename>: show changes you made relative to the staging area
- git diff <revision> <filename>: shows differences in a file between snapshots
- git checkout <revision>: updates HEAD and current branch

用法:看介绍



Scenario-1: work on a local project [2]

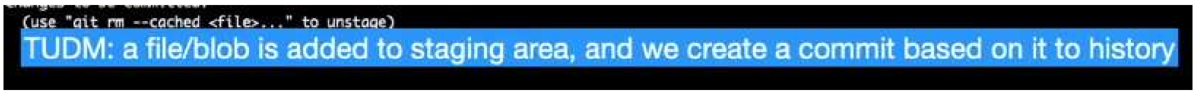
```
┌─dd@dd-PC7 ~/devlop/git-tutorial ‹main›
└─$ echo "hello git" >> hello.txt
┌─dd@dd-PC7 ~/devlop/git-tutorial ‹main*›
└─$ ls
hello.txt
┌─dd@dd-PC7 ~/devlop/git-tutorial ‹main*›
└─$ git status
On branch main

No commits yet

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

nothing added to commit but untracked files present (use "git add" to track)
┌─dd@dd-PC7 ~/devlop/git-tutorial ‹main*›
└─$ git add hello.txt
┌─dd@dd-PC7 ~/devlop/git-tutorial ‹main*›
└─$ git status
On branch main

No commits yet

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

```
┌─dd@dd-PC7 ~/devlop/git-tutorial ‹main*›
└─$ git commit -m "init commit"
[main (root-commit) 58936ec] init commit
 1 file changed, 1 insertion(+)
 create mode 100644 hello.txt
┌─dd@dd-PC7 ~/devlop/git-tutorial ‹main›
└─$ git status
On branch main
nothing to commit, working tree clean
```

Scenario场景

echo写入命令

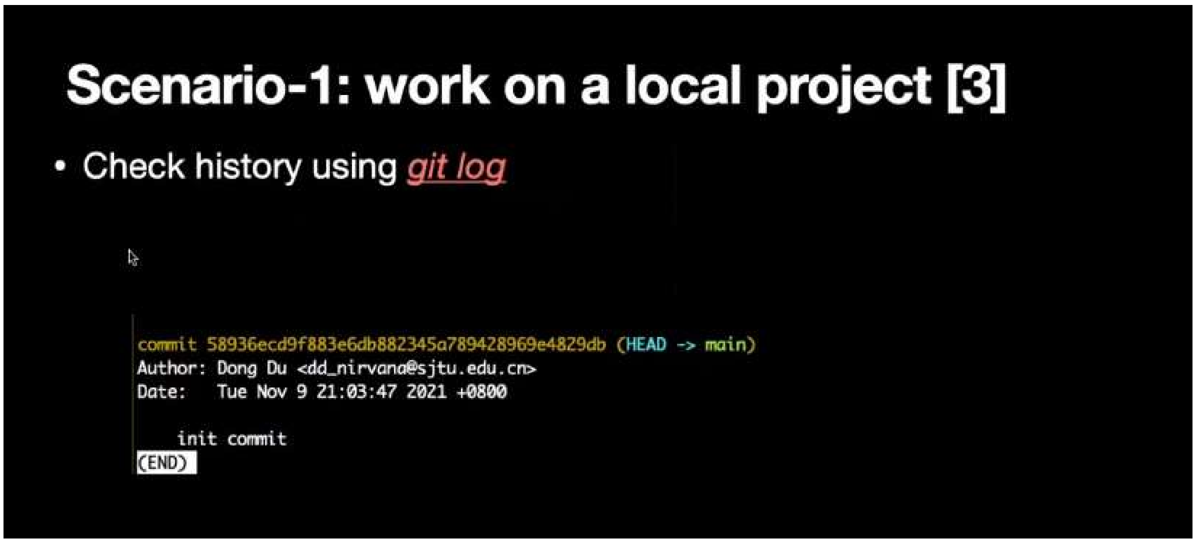
git commit创建一个新的snapshot



```
  (use "git rm --cached <file>..." to unstage)
```

TUDM: a file/blob is added to staging area, and we create a commit based on it to history

查看日志

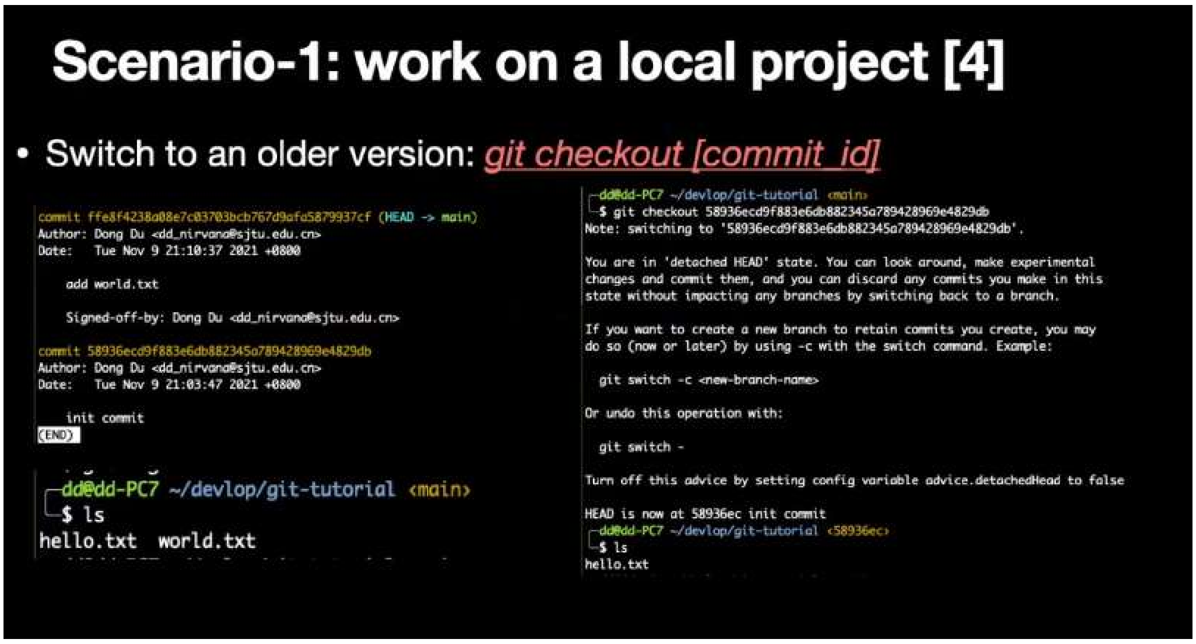


Scenario-1: work on a local project [3]

- Check history using *git log*

```
commit 58936ecd9f883e6db882345a789428969e4829db (HEAD -> main)
Author: Dong Du <dd_nirvana@sjtu.edu.cn>
Date:   Tue Nov 9 21:03:47 2021 +0800

    init commit
(END)
```

切换版本，就是当前改烂了，切换到原来版本



Scenario-1: work on a local project [4]

- Switch to an older version: *git checkout [commit_id]*

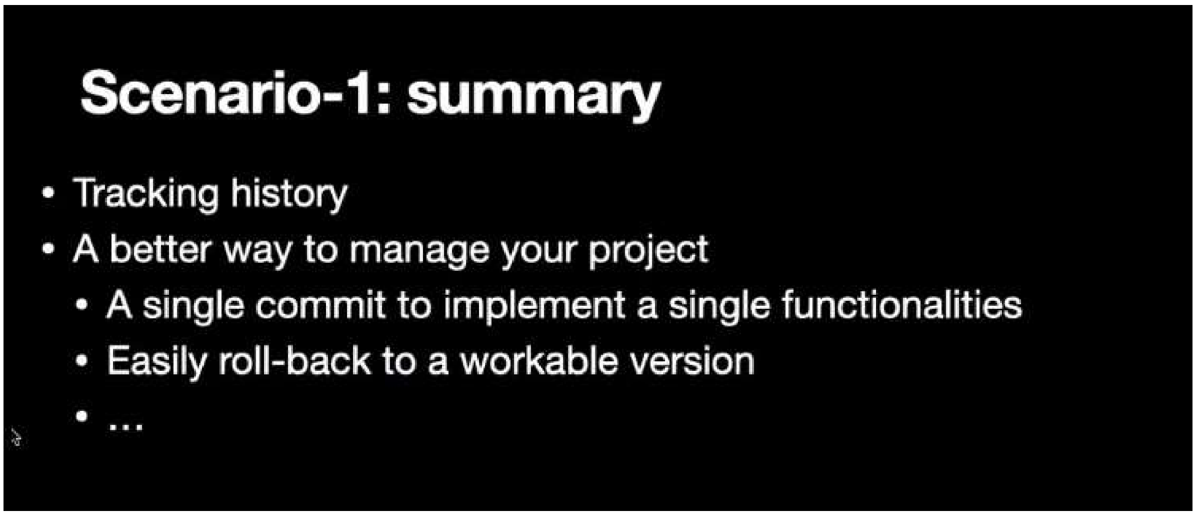
```
commit ffe8f4238a08e7c03703bcb767d9afa5879937cf (HEAD -> main)
Author: Dong Du <dd_nirvana@sjtu.edu.cn>
Date:   Tue Nov 9 21:10:37 2021 +0800

    add world.txt

    Signed-off-by: Dong Du <dd_nirvana@sjtu.edu.cn>

commit 58936ecd9f883e6db882345a789428969e4829db
Author: Dong Du <dd_nirvana@sjtu.edu.cn>
Date:   Tue Nov 9 21:03:47 2021 +0800

    init commit
(END)
```

```
┌─dd@dd-PC7 ~/devlop/git-tutorial <main>
└─$ ls
hello.txt  world.txt
```

```
┌─dd@dd-PC7 ~/devlop/git-tutorial <main>
└─$ git checkout 58936ecd9f883e6db882345a789428969e4829db
Note: switching to '58936ecd9f883e6db882345a789428969e4829db'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>

Or undo this operation with:

  git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at 58936ec init commit
┌─dd@dd-PC7 ~/devlop/git-tutorial <58936ec>
└─$ ls
hello.txt
```

回到原来的版本的方法，后面加版本哈希码

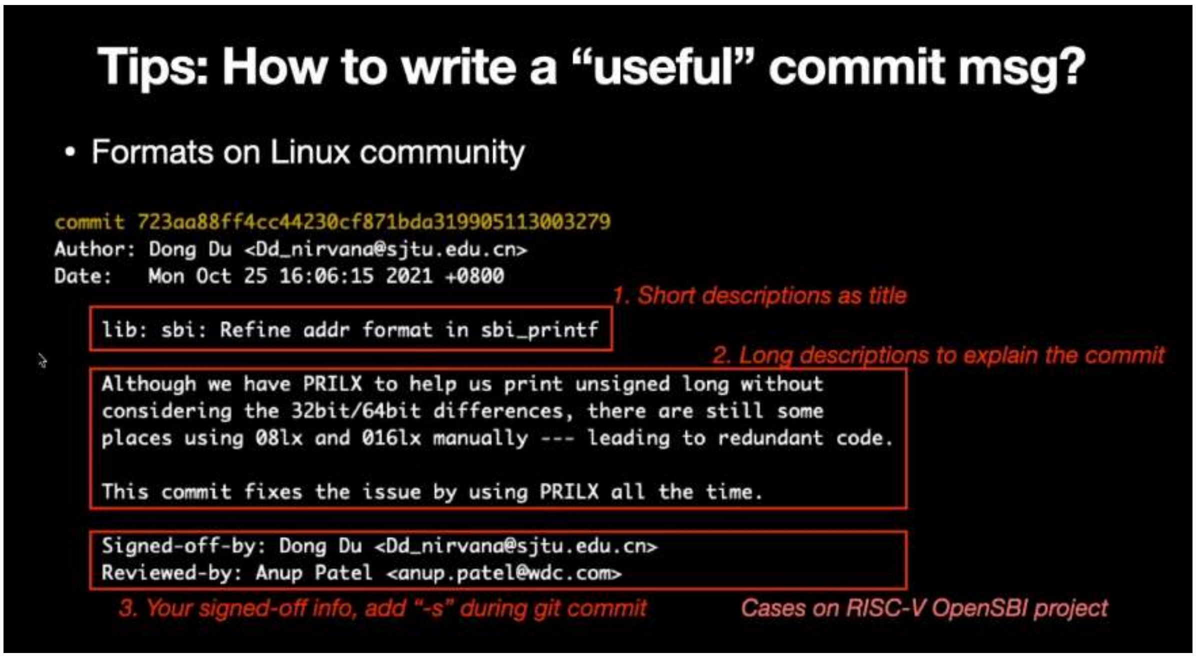
git diff查看有什么修改



Scenario-1: summary

- Tracking history
- A better way to manage your project
  - A single commit to implement a single functionalities
  - Easily roll-back to a workable version
  - …

commit msg(message)的书写



Tips: How to write a “useful” commit msg?

- Formats on Linux community

```
commit 723aa88ff4cc44230cf871bda319905113003279
Author: Dong Du <Dd_nirvana@sjtu.edu.cn>
Date:   Mon Oct 25 16:06:15 2021 +0800

    lib: sbi: Refine addr format in sbi_printf

    Although we have PRILX to help us print unsigned long without
    considering the 32bit/64bit differences, there are still some
    places using 08lx and 016lx manually --- leading to redundant code.

    This commit fixes the issue by using PRILX all the time.

    Signed-off-by: Dong Du <Dd_nirvana@sjtu.edu.cn>
    Reviewed-by: Anup Patel <anup.patel@wdc.com>
```

1. Short descriptions as title

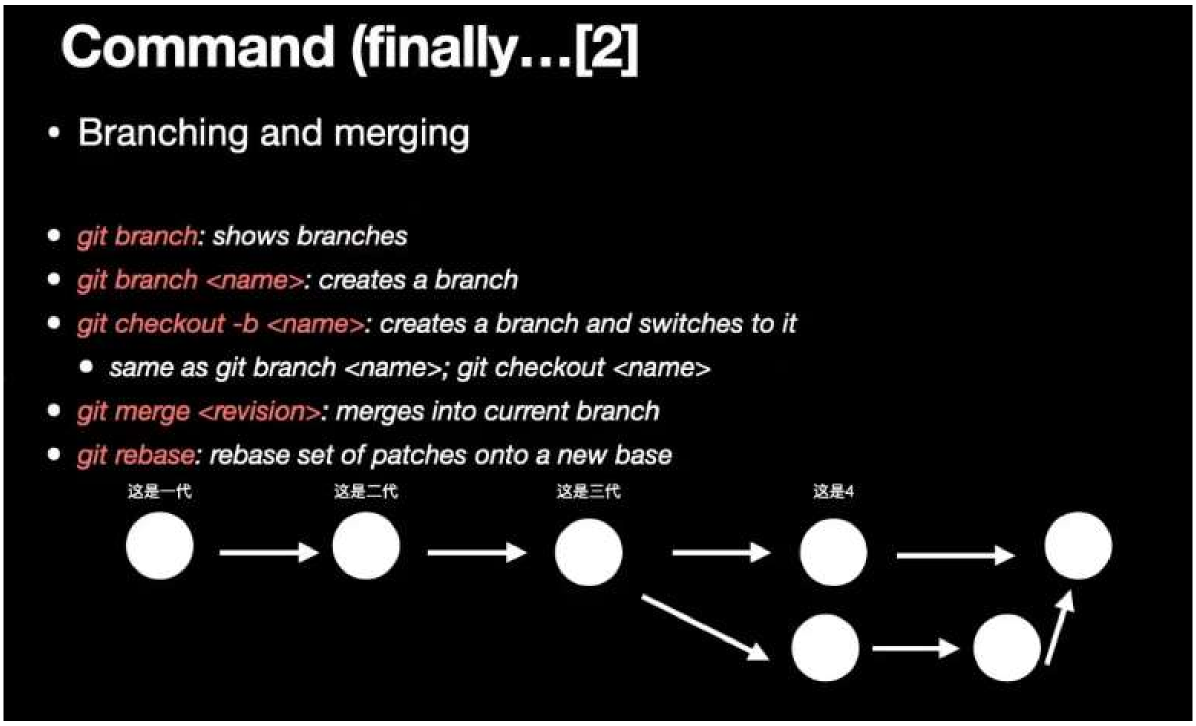
2. Long descriptions to explain the commit

3. Your signed-off info, add “-s” during git commit

Cases on RISC-V OpenSBI project

按上面格式写

branching创建分支和merge分支



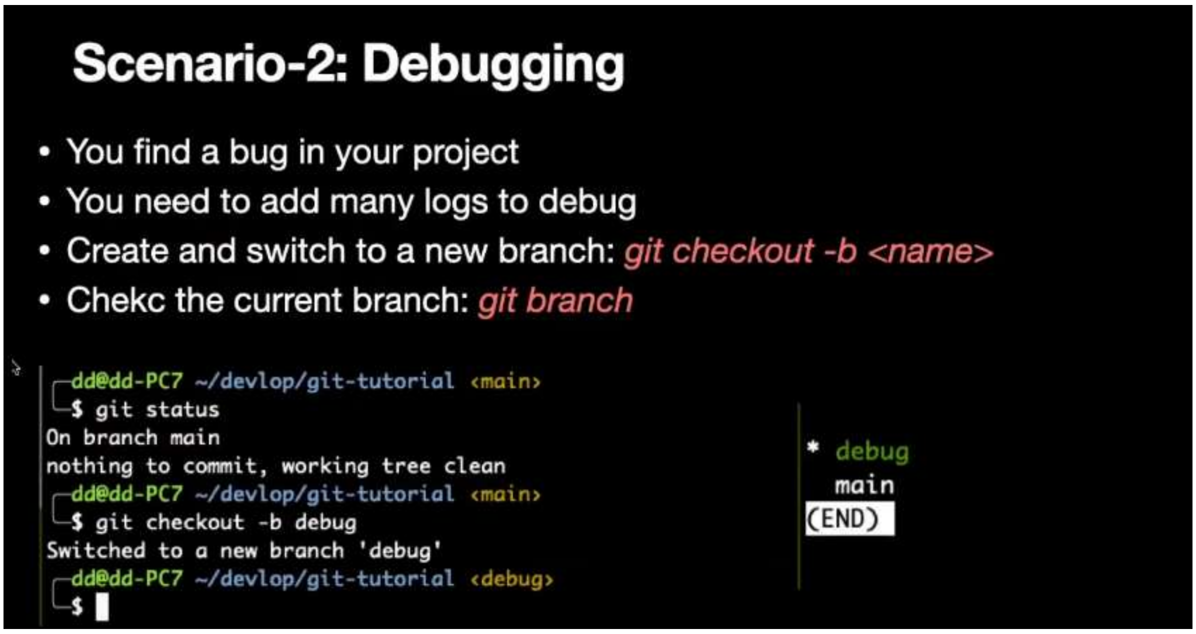
Command (finally…[2]

- Branching and merging

- *git branch: shows branches*
- *git branch <name>: creates a branch*
- *git checkout -b <name>: creates a branch and switches to it*
  - *same as git branch <name>; git checkout <name>*
- *git merge <revision>: merges into current branch*
- *git rebase: rebase set of patches onto a new base*

这是一代
这是二代
这是三代
这是4

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*

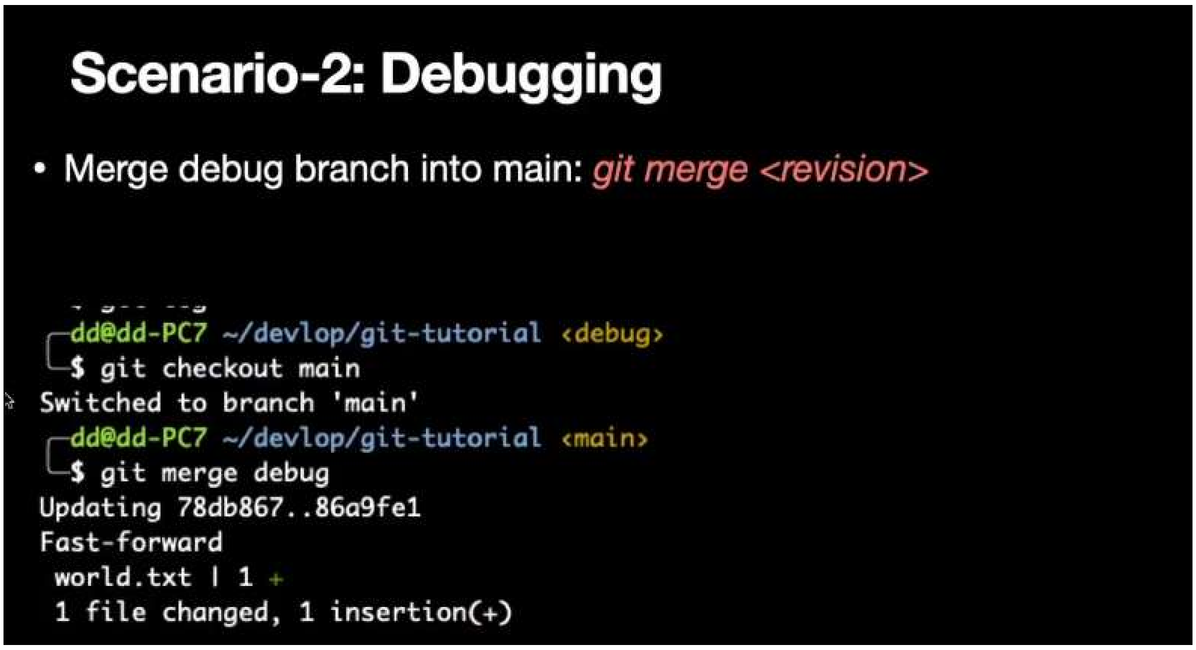
```
┌dd@dd-PC7 ~/devlop/git-tutorial ‹main›
└$ git status
On branch main
nothing to commit, working tree clean
┌dd@dd-PC7 ~/devlop/git-tutorial ‹main›
└$ git checkout -b debug
Switched to a new branch 'debug'
┌dd@dd-PC7 ~/devlop/git-tutorial ‹debug›
└$
```

```
* debug
  main
(END)
```

git checkout -b XXX创建一个分支

git branch

合并分支merge



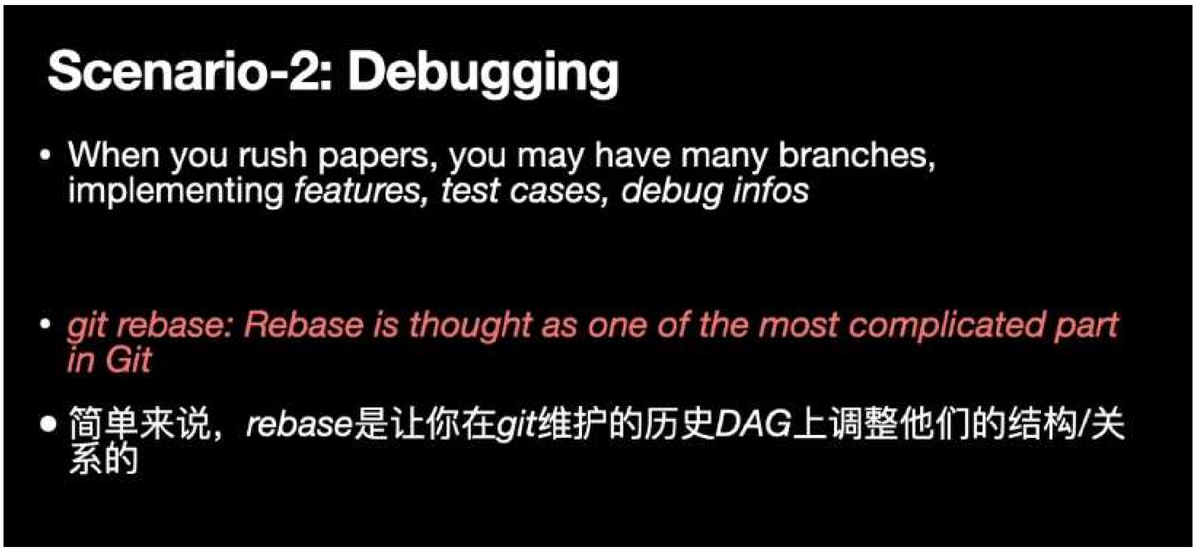
Scenario-2: Debugging

- 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

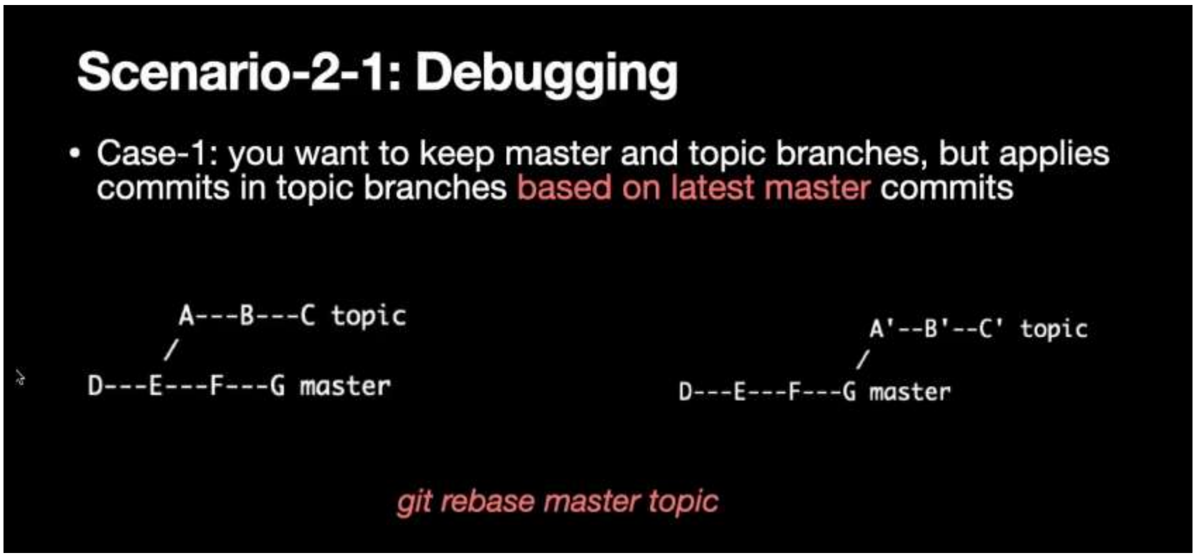
！！！！！合并多个不同分支



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

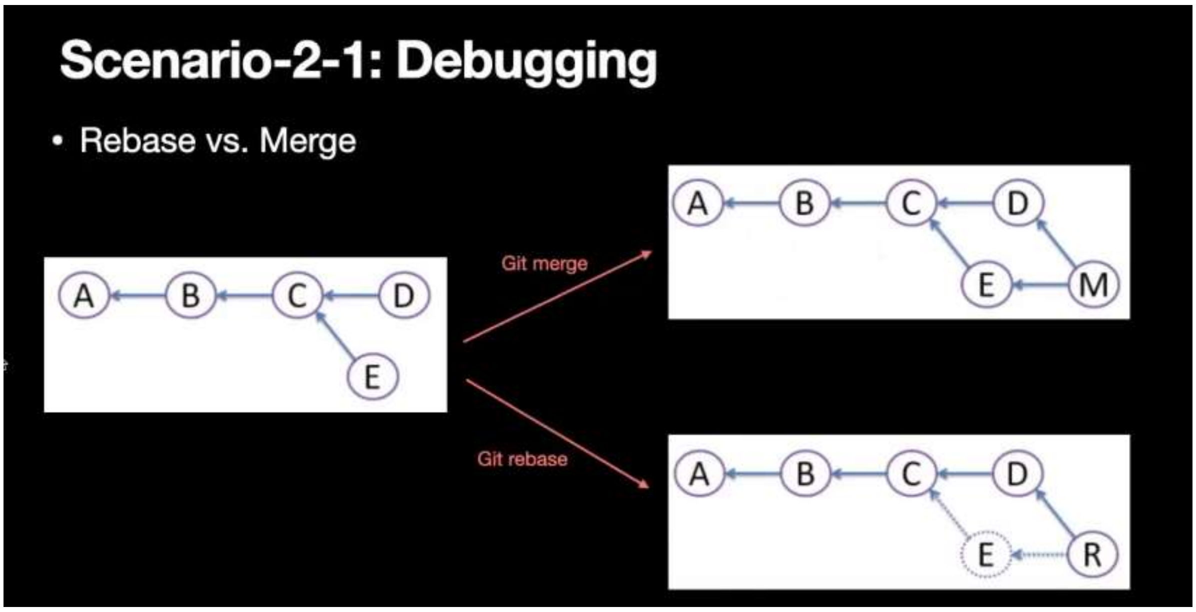
```
          A---B---C topic
         /
    D---E---F---G master
```

```
                  A'--B'--C' topic
                 /
    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）

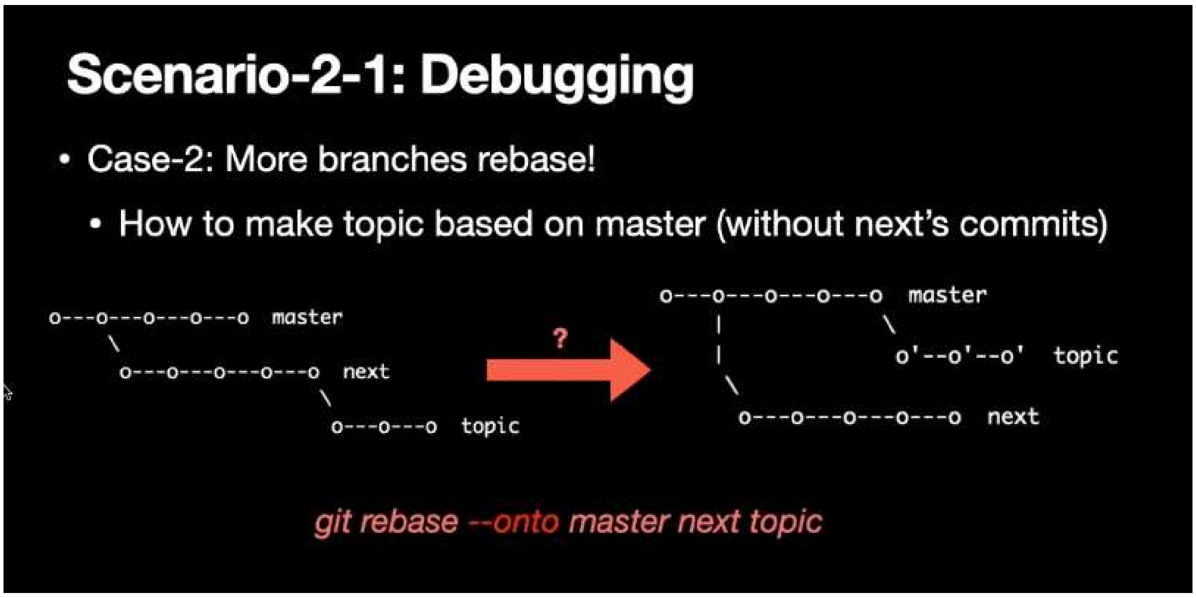


Scenario-2-1: Debugging

- 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：



Scenario-2-1: Debugging

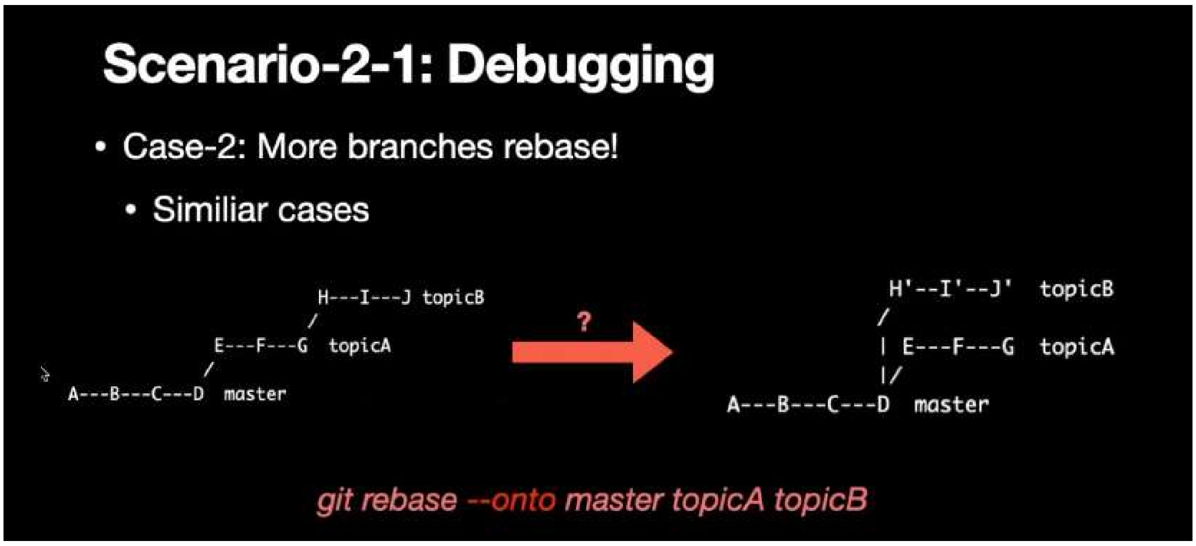
- Case-2: More branches rebase!
  - How to make topic based on master (without next's commits)

```
o---o---o---o---o  master
     \
      o---o---o---o---o  next
                       \
                        o---o---o  topic
```

```
o---o---o---o---o  master
    |            \
    |             o'--o'--o'  topic
     \
      o---o---o---o---o  next
```

git rebase --onto master next topic

onto相当于加个reference把从next到topic的干过去



Scenario-2-1: Debugging

- Case-2: More branches rebase!
  - Similiar cases

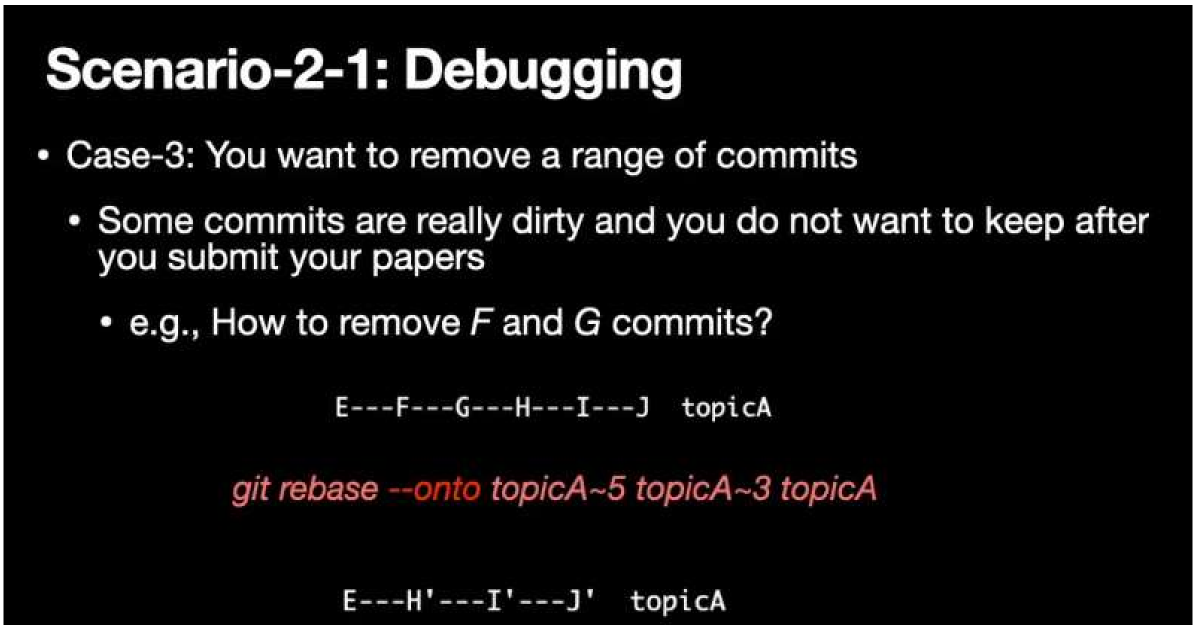
```
                        H---I---J topicB
                       /
              E---F---G  topicA
             /
A---B---C---D  master
```

```
               H'--I'--J'  topicB
              /
             | E---F---G  topicA
             |/
A---B---C---D  master
```

git rebase --onto master topicA topicB

更改区间位置

骚操作，利用git rebase删掉几个commit



Scenario-2-1: Debugging

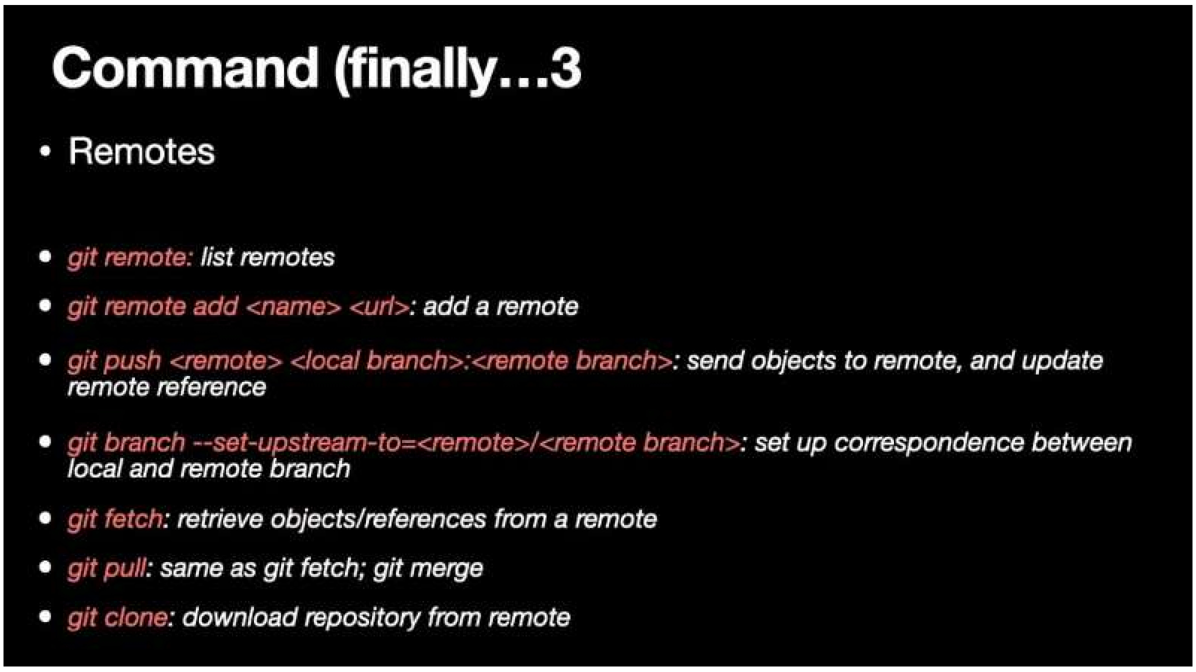
- Case-3: You want to remove a range of commits
  - Some commits are really dirty and you do not want to keep after you submit your papers
    - e.g., How to remove *F* and *G* commits?

```
E---F---G---H---I---J  topicA
```

git rebase --onto topicA~5 topicA~3 topicA

```
E---H'---I'---J'  topicA
```

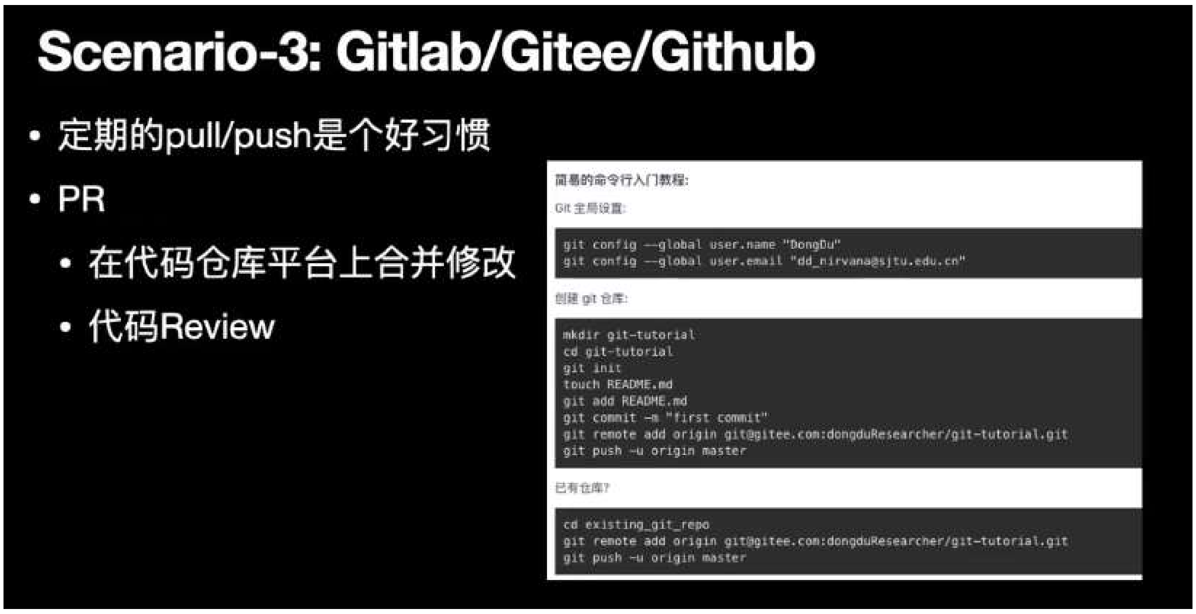
Remote（遥控，偏远的）就是具体拉取上传代码的操作



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"
```

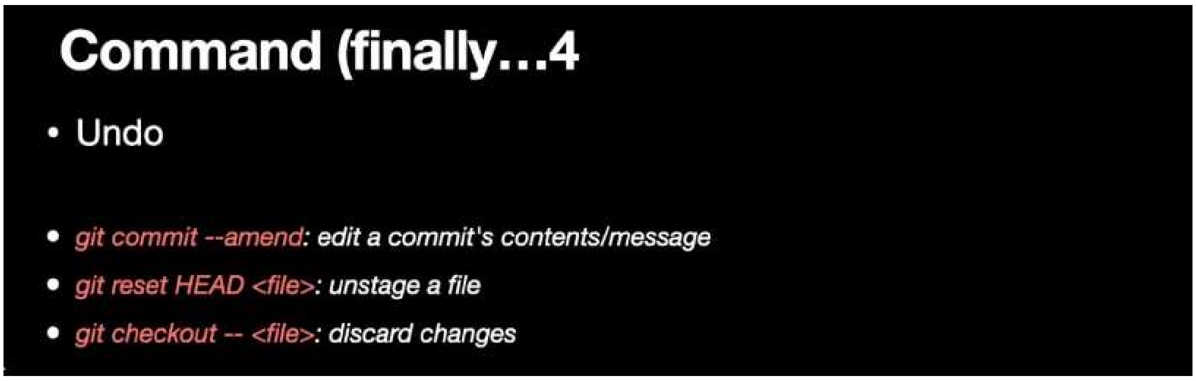
创建 git 仓库:

```
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
```

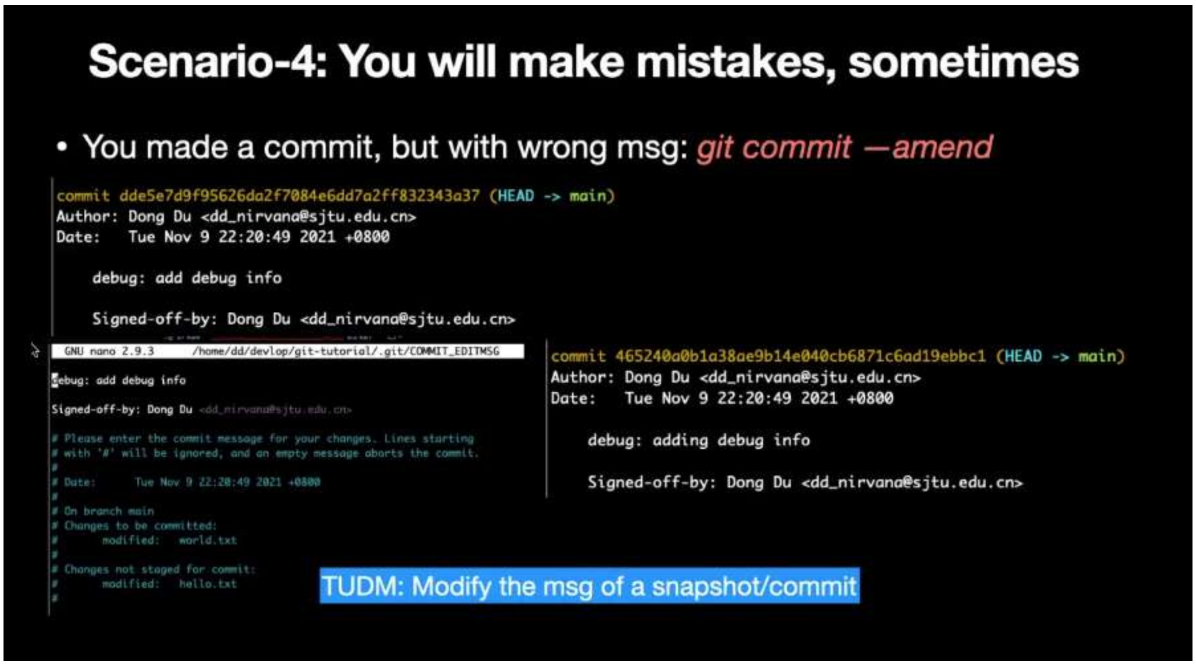
Undo撤销相关的操作



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



Scenario-4: You will make mistakes, sometimes

- You made a commit, but with wrong msg: *git commit —amend*

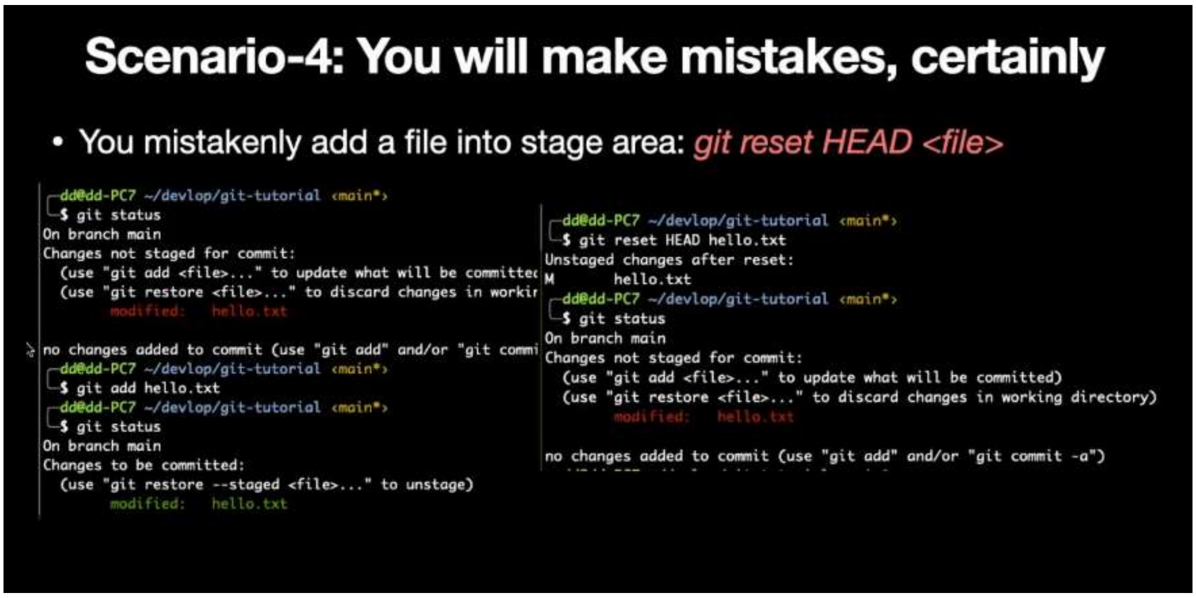
```
commit dde5e7d9f95626da2f7084e6dd7a2ff832343a37 (HEAD -> main)
Author: Dong Du <dd_nirvana@sjtu.edu.cn>
Date:   Tue Nov 9 22:20:49 2021 +0800

    debug: add debug info

    Signed-off-by: Dong Du <dd_nirvana@sjtu.edu.cn>
```

```
 GNU nano 2.9.3      /home/dd/devlop/git-tutorial/.git/COMMIT_EDITMSG

debug: add debug info

Signed-off-by: Dong Du <dd_nirvana@sjtu.edu.cn>

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date:      Tue Nov 9 22:20:49 2021 +0800
#
# On branch main
# Changes to be committed:
#       modified:   world.txt
#
# Changes not staged for commit:
#       modified:   hello.txt
#
```

```
commit 465240a0b1a38ae9b14e040cb6871c6ad19ebbc1 (HEAD -> main)
Author: Dong Du <dd_nirvana@sjtu.edu.cn>
Date:   Tue Nov 9 22:20:49 2021 +0800

    debug: adding debug info

    Signed-off-by: Dong Du <dd_nirvana@sjtu.edu.cn>
```

TUDM: Modify the msg of a snapshot/commit

修改上个commit（--amend有俩杠）



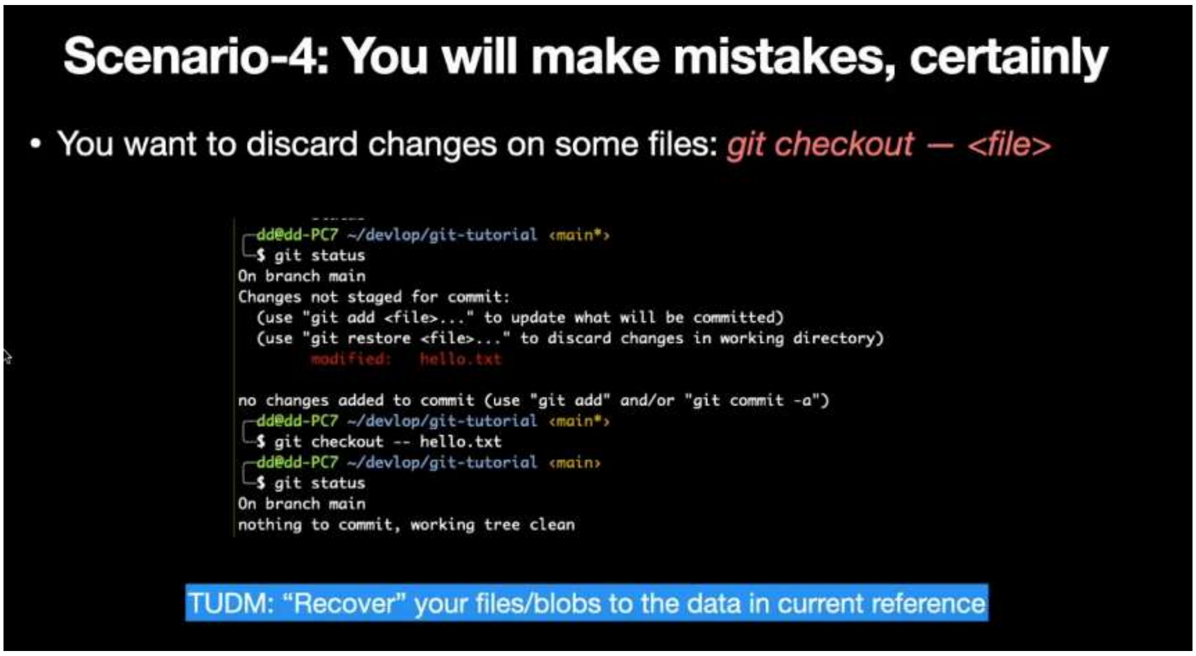
Scenario-4: You will make mistakes, certainly

- You mistakenly add a file into stage area: *git reset HEAD <file>*

```
┌dd@dd-PC7 ~/devlop/git-tutorial ‹main*›
└$ git status
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committec
  (use "git restore <file>..." to discard changes in workir
        modified:   hello.txt

no changes added to commit (use "git add" and/or "git commi
┌dd@dd-PC7 ~/devlop/git-tutorial ‹main*›
└$ git add hello.txt
┌dd@dd-PC7 ~/devlop/git-tutorial ‹main*›
└$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   hello.txt
```

```
┌dd@dd-PC7 ~/devlop/git-tutorial ‹main*›
└$ git reset HEAD hello.txt
Unstaged changes after reset:
M       hello.txt
┌dd@dd-PC7 ~/devlop/git-tutorial ‹main*›
└$ git status
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   hello.txt

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

修改stage area里的文件但是保存



Scenario-4: You will make mistakes, certainly

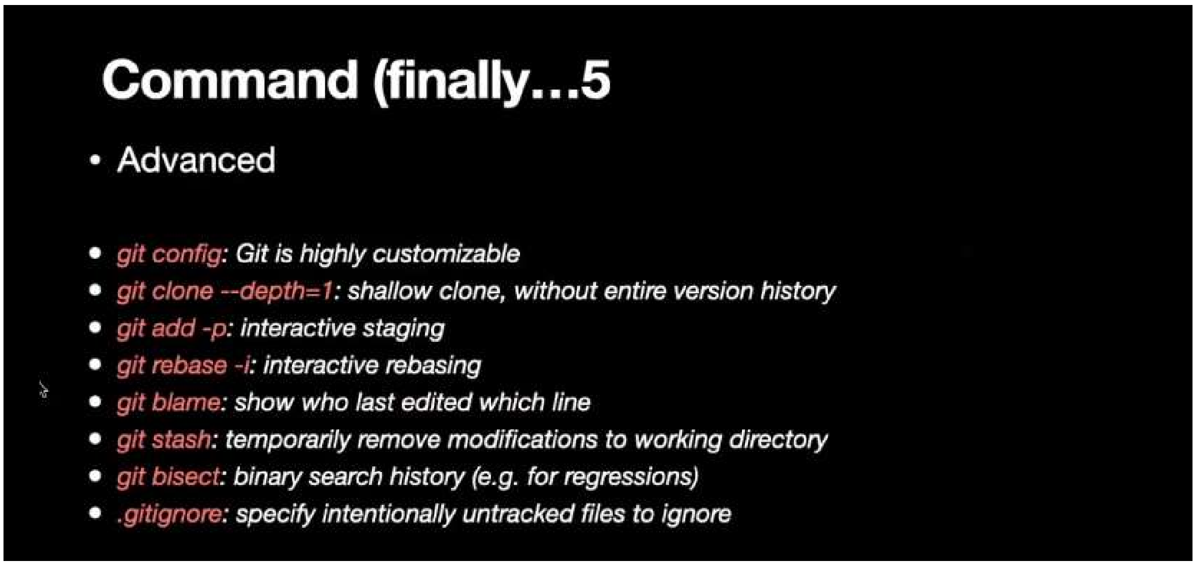
- You want to discard changes on some files: *git checkout — <file>*

```
┌dd@dd-PC7 ~/devlop/git-tutorial ‹main*›
└$ git status
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   hello.txt

no changes added to commit (use "git add" and/or "git commit -a")
┌dd@dd-PC7 ~/devlop/git-tutorial ‹main*›
└$ git checkout -- hello.txt
┌dd@dd-PC7 ~/devlop/git-tutorial ‹main›
└$ git status
On branch main
nothing to commit, working tree clean
```

TUDM: “Recover” your files/blobs to the data in current reference

彻底删掉stage area里的有修改动作文件

高级操作

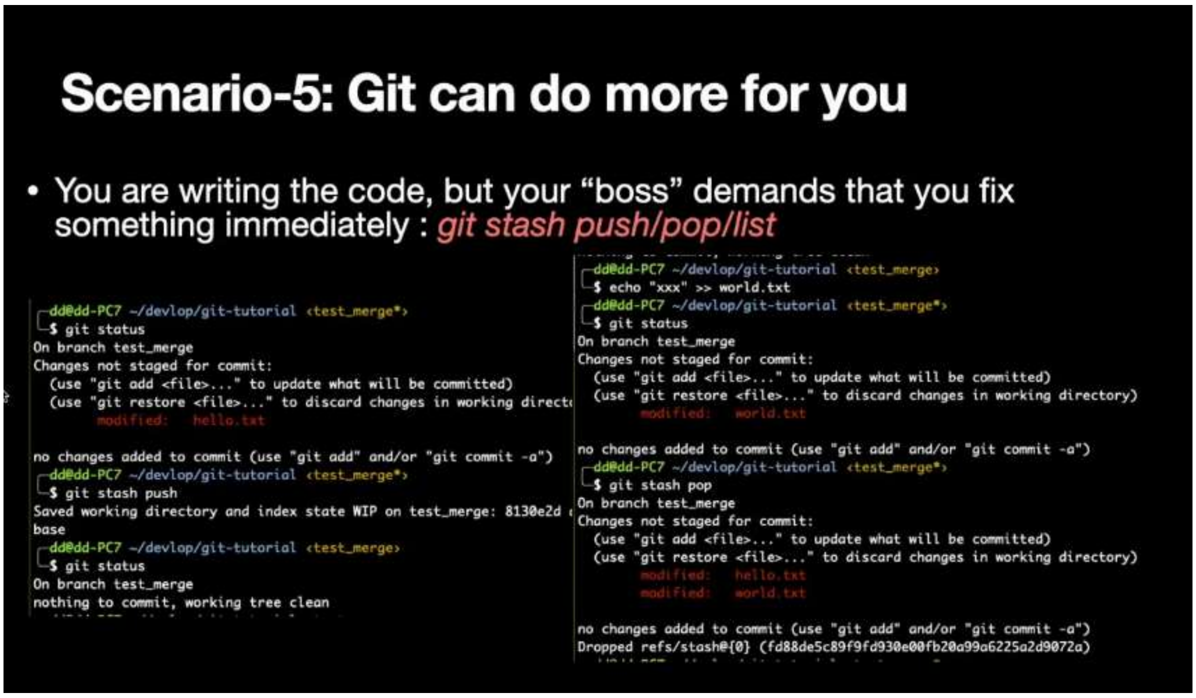


Command (finally…5

- Advanced
  - *git config: Git is highly customizable*
  - *git clone --depth=1: shallow clone, without entire version history*
  - *git add -p: interactive staging*
  - *git rebase -i: interactive rebasing*
  - *git blame: show who last edited which line*
  - *git stash: temporarily remove modifications to working directory*
  - *git bisect: binary search history (e.g. for regressions)*
  - *.gitignore: specify intentionally untracked files to ignore*

git blame，查某个改动是谁做的

git stash push/pop/list



Scenario-5: Git can do more for you

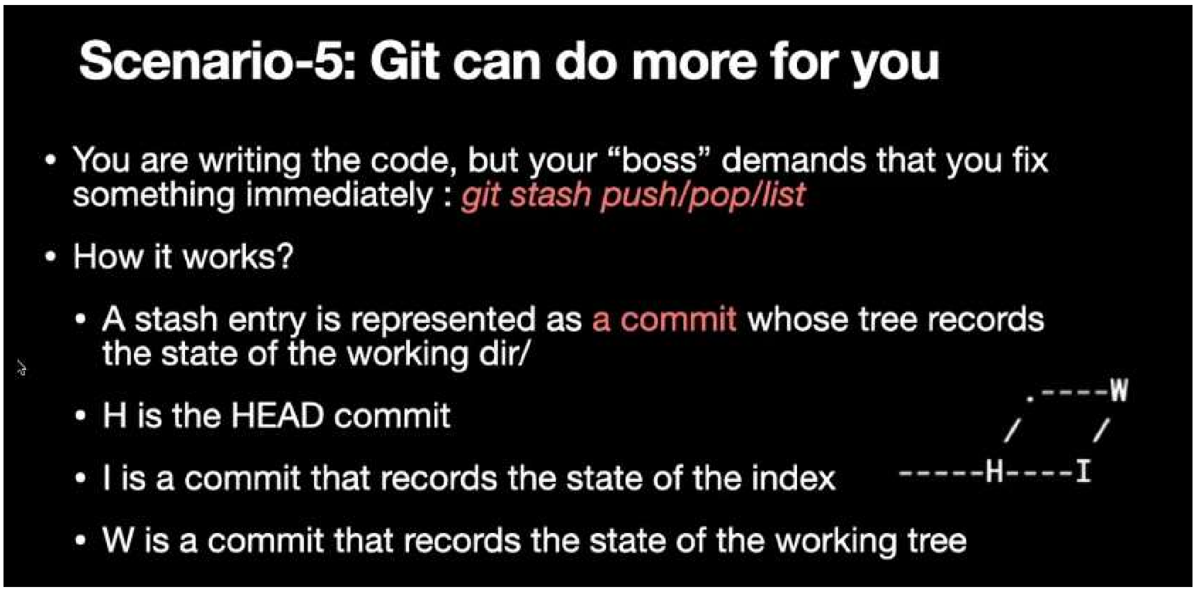
- You are writing the code, but your “boss” demands that you fix something immediately : *git stash push/pop/list*

```
┌dd@dd-PC7 ~/devlop/git-tutorial ‹test_merge*›
└$ git status
On branch test_merge
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directc
        modified:   hello.txt

no changes added to commit (use "git add" and/or "git commit -a")
┌dd@dd-PC7 ~/devlop/git-tutorial ‹test_merge*›
└$ git stash push
Saved working directory and index state WIP on test_merge: 8130e2d
base
┌dd@dd-PC7 ~/devlop/git-tutorial ‹test_merge›
└$ git status
On branch test_merge
nothing to commit, working tree clean
```

```
┌dd@dd-PC7 ~/devlop/git-tutorial ‹test_merge›
└$ echo "xxx" >> world.txt
┌dd@dd-PC7 ~/devlop/git-tutorial ‹test_merge*›
└$ git status
On branch test_merge
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   world.txt

no changes added to commit (use "git add" and/or "git commit -a")
┌dd@dd-PC7 ~/devlop/git-tutorial ‹test_merge*›
└$ git stash pop
On branch test_merge
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   hello.txt
        modified:   world.txt

no changes added to commit (use "git add" and/or "git commit -a")
Dropped refs/stash@{0} (fd88de5c89f9fd930e00fb20a99a6225a2d9072a)
```

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

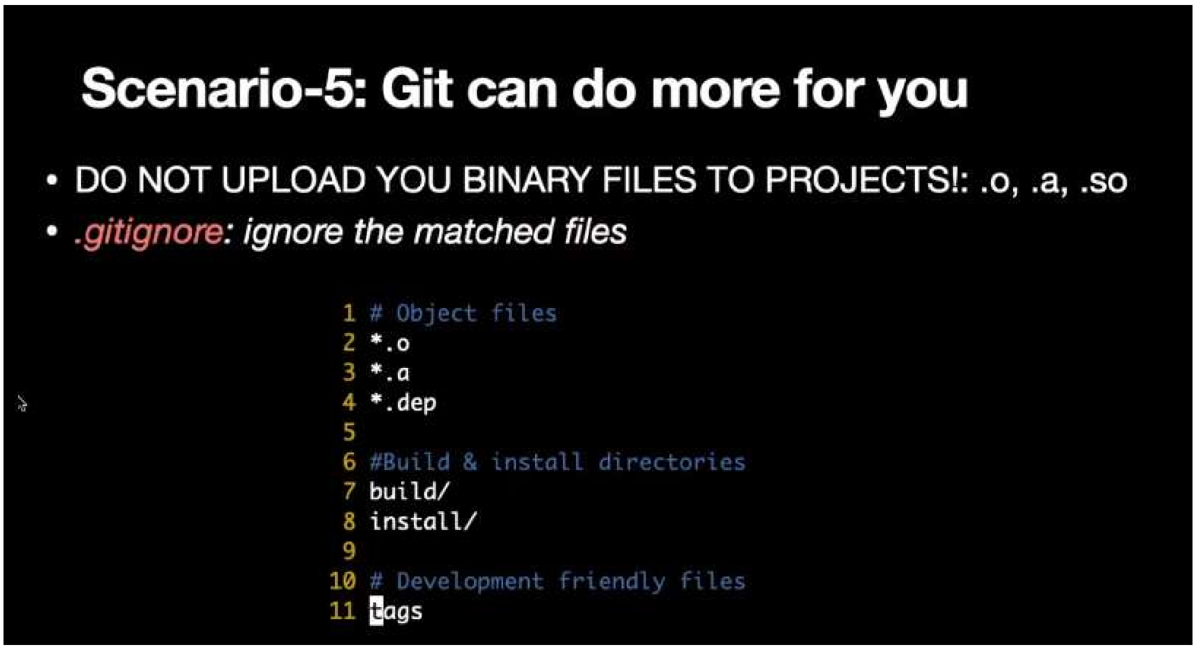


Scenario-5: Git can do more for you

- 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*

```
# Object files
*.o
*.a
*.dep

#Build & install directories
build/
install/

# Development friendly files
tags
```