IPADS Tutorial - Git

Dong Du, IPADS, 2021-11-10, ***some contents from MIT's missing-semester***

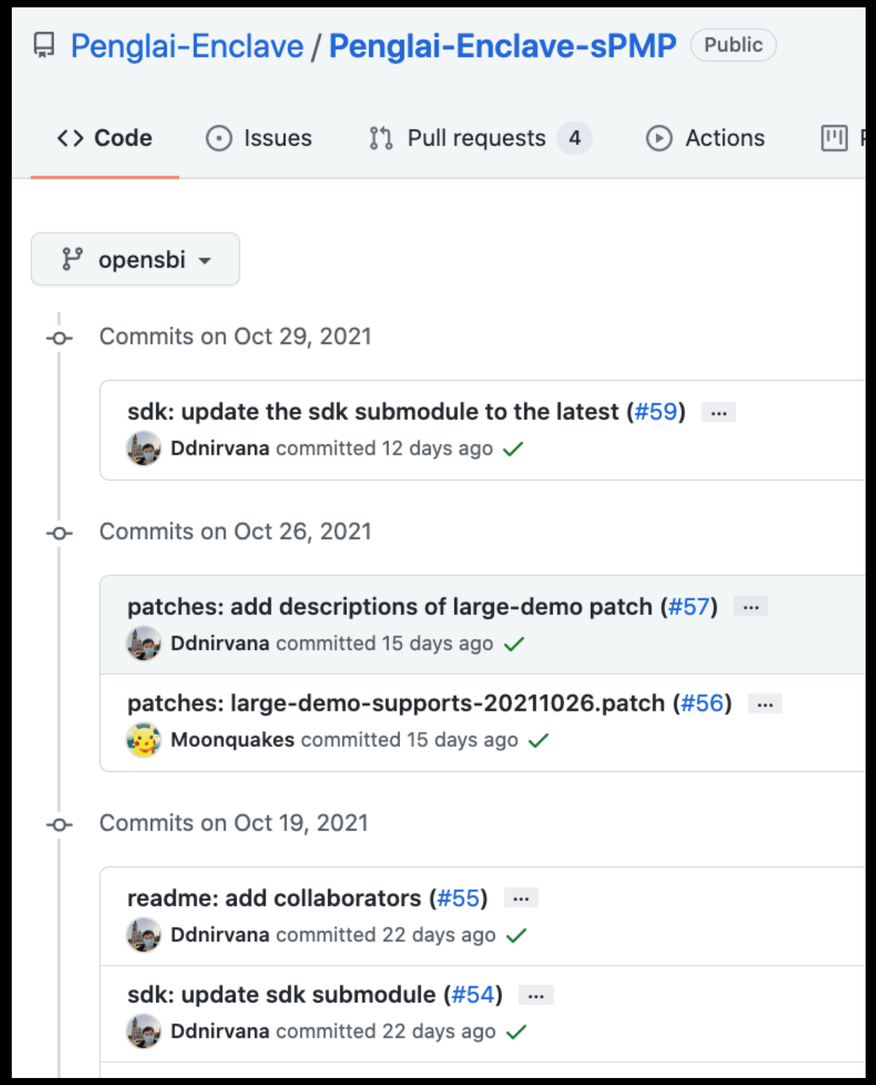
Outline

- Background
- Data models
- Commands
- Case Studies

Version Control System（版本控制）

- *Git (/gɪt/) is software for tracking changes in any set of files, usually used for coordinating work among programmers collaboratively developing source code during software development. Its goals include speed, data integrity, and support for distributed, non-linear workflows (thousands of parallel branches running on different systems) -- Wikipedia*

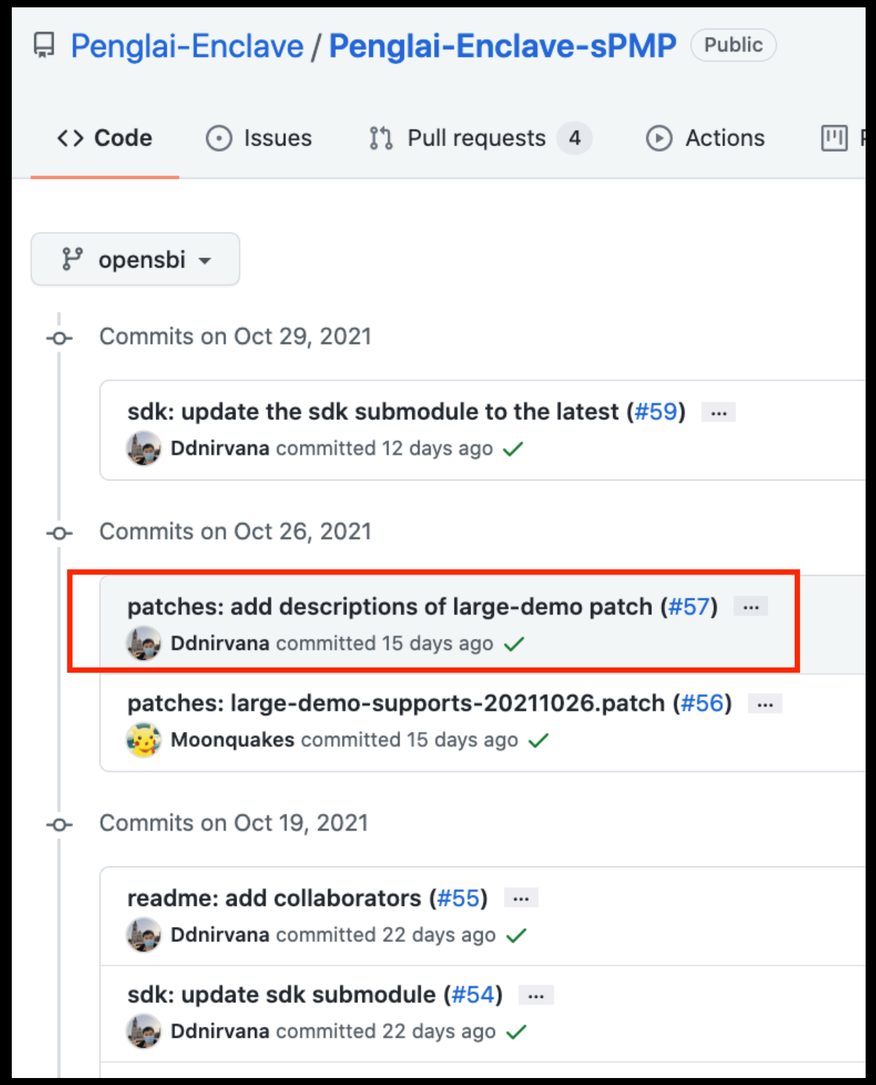
Version Control System（版本控制）



History (many versions)

Changes (in a specific version)

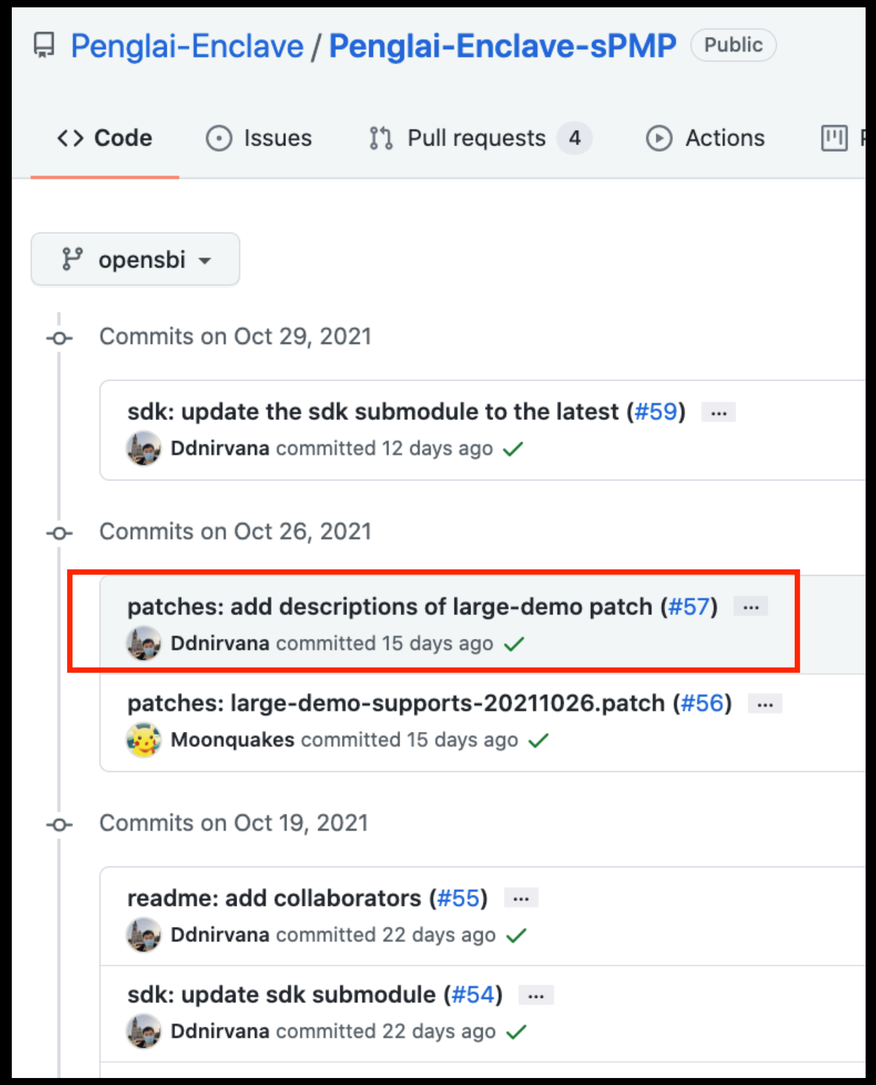
Version Control System（版本控制）



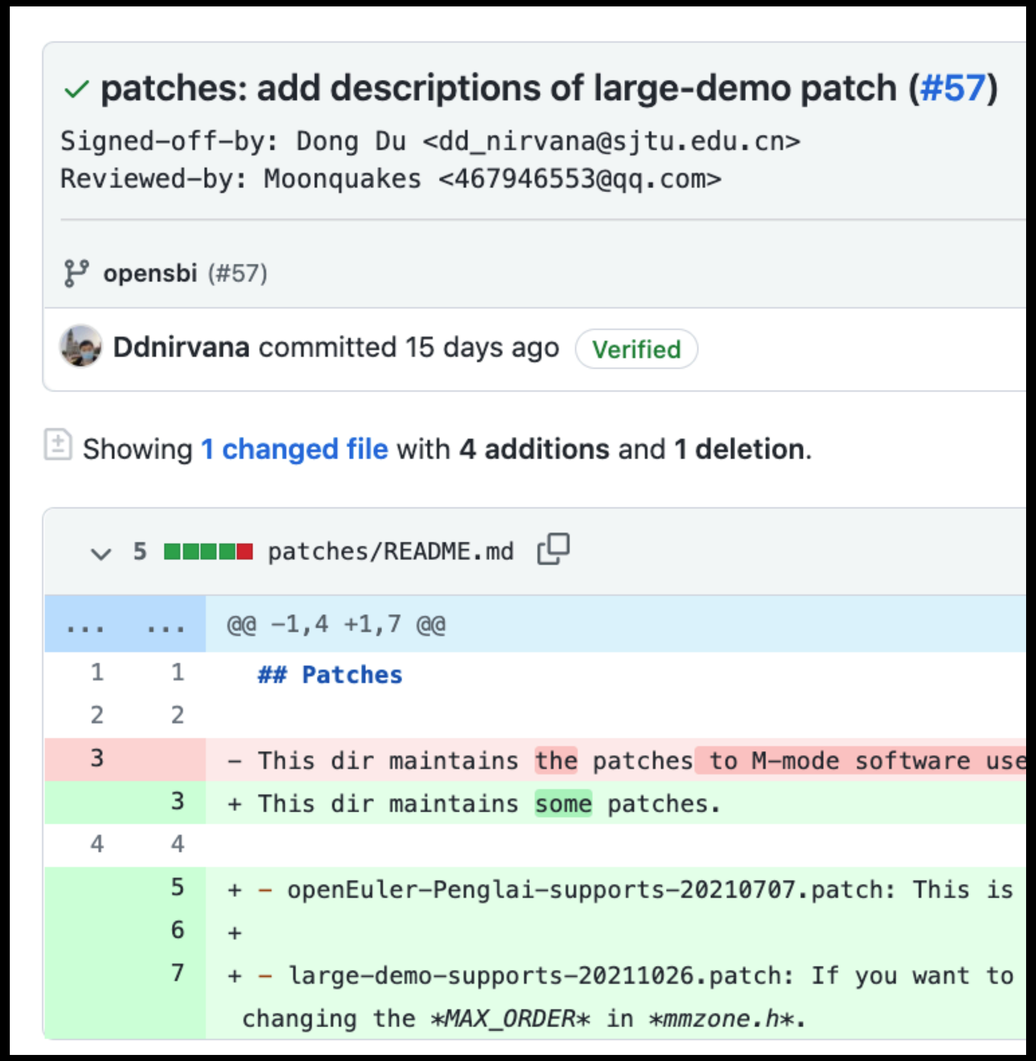
History (many versions)

Changes (in a specific version)

Version Control System（版本控制）

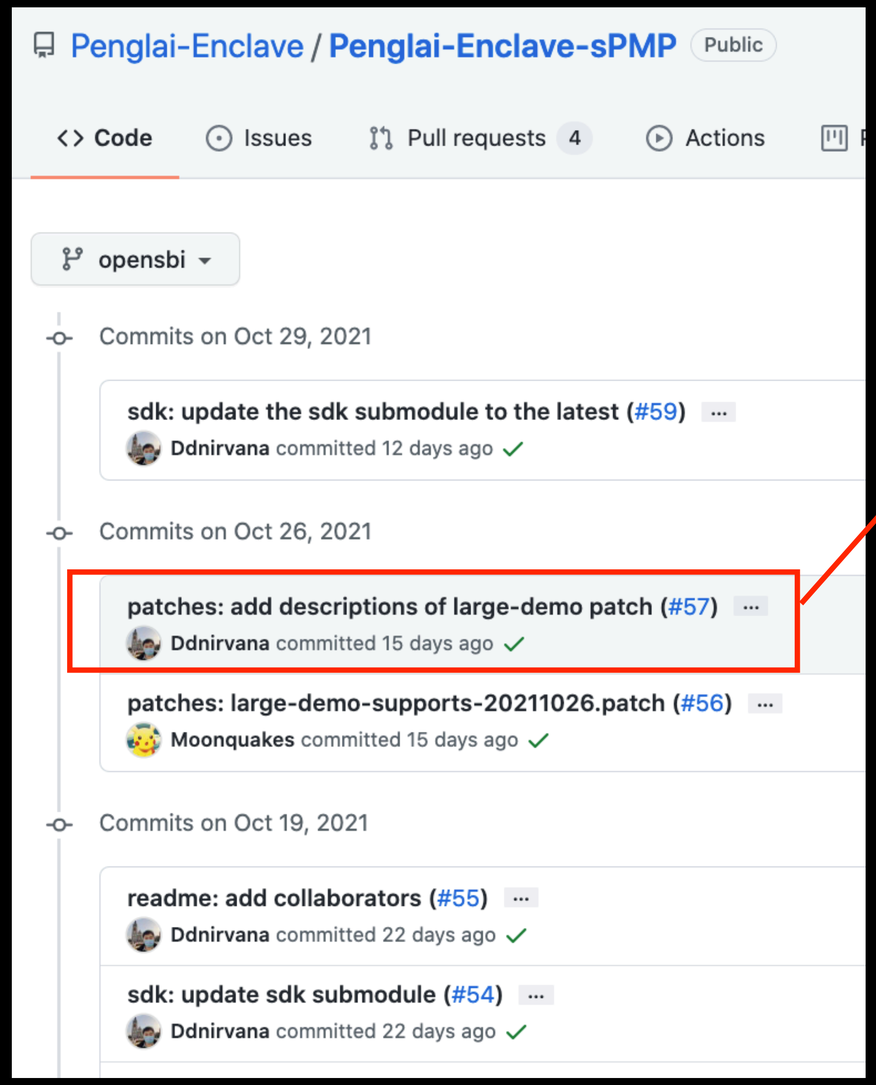


History (many versions)

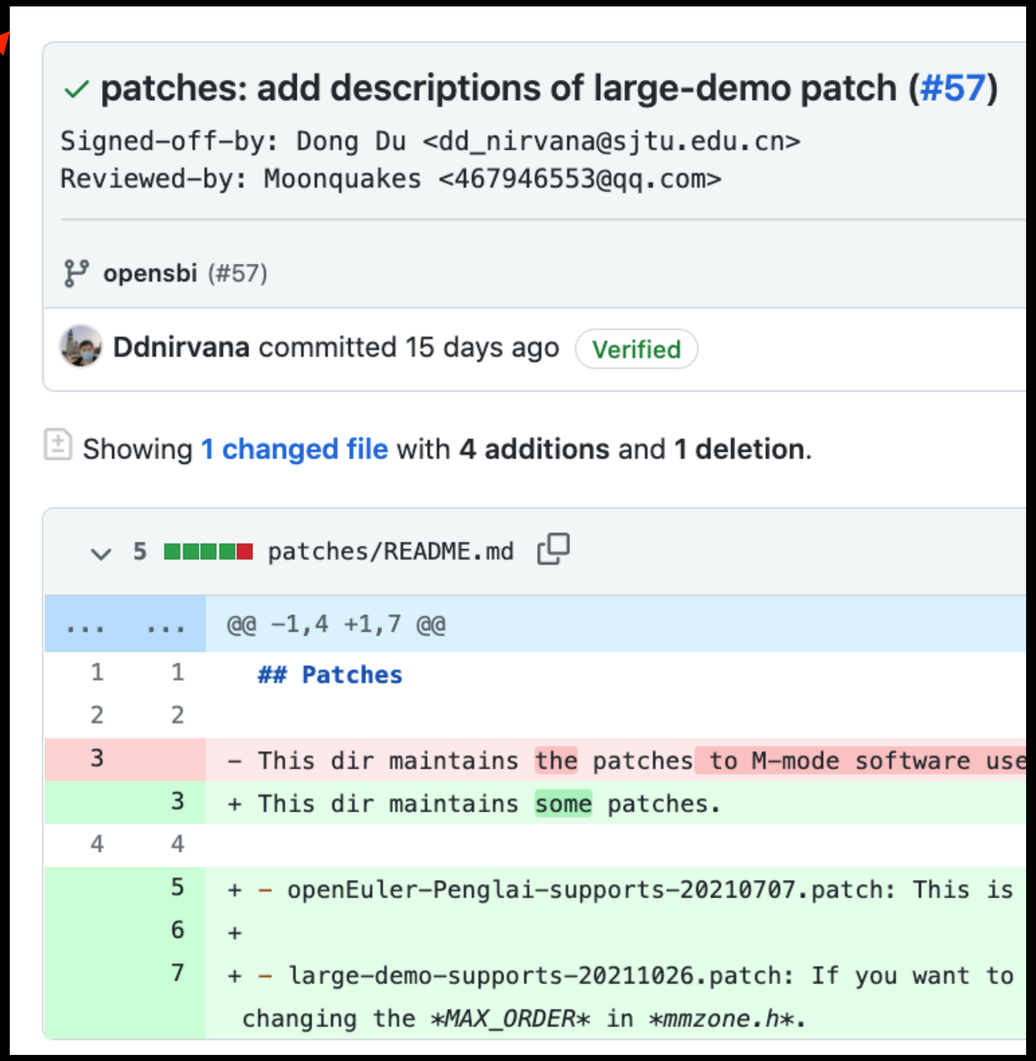


Changes (in a specific version)

Version Control System（版本控制）



History (many versions)



Changes (in a specific version)

Why version control?

- *Working by yourself*
  - *Look at old versions of a project*
  - *Keep a log of why certain changes were made*
  - *Work on parallel branches of development*
- *Working with others*
  - *See what other people have changed, learn and review*
  - *Resolve conflicts in concurrent development*

How to “learn” Git?

How to “learn” Git?

How to “learn” Git?

Just memorize shell commands?

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*

Thinking of history: story of snapshots

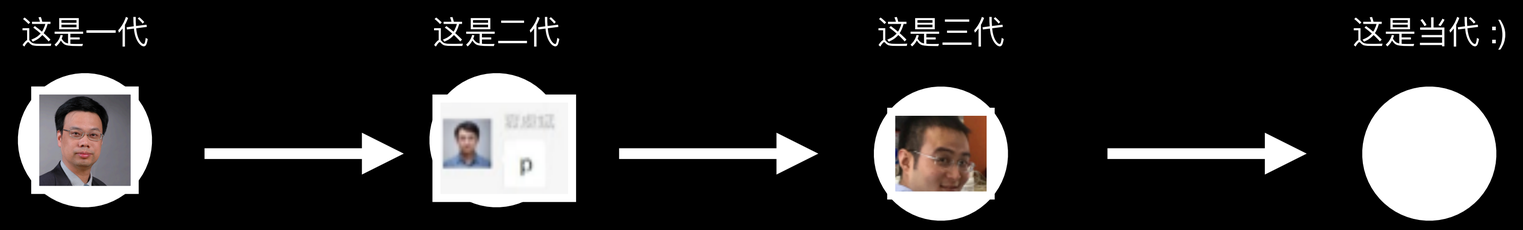
- // Skip the definition of snapshots now

Thinking of history: story of snapshots

- // Skip the definition of snapshots now
- Simple model: linear history?
  - A list of snapshots in time-order

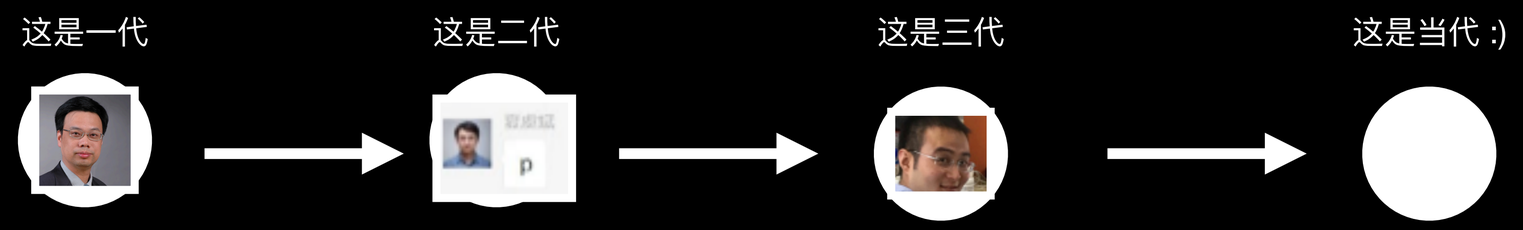
Thinking of history: story of snapshots

- // Skip the definition of snapshots now
- Simple model: linear history?
  - A list of snapshots in time-order



Thinking of history: story of snapshots

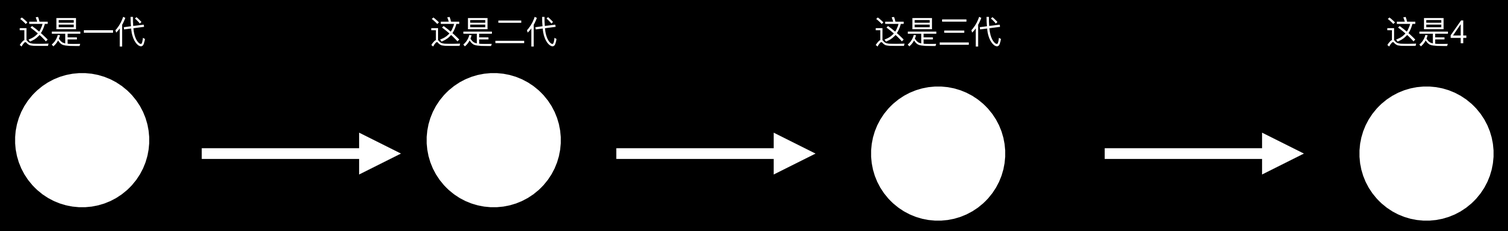
- // Skip the definition of snapshots now
- Simple model: linear history?
  - A list of snapshots in time-order



Git does not use this model

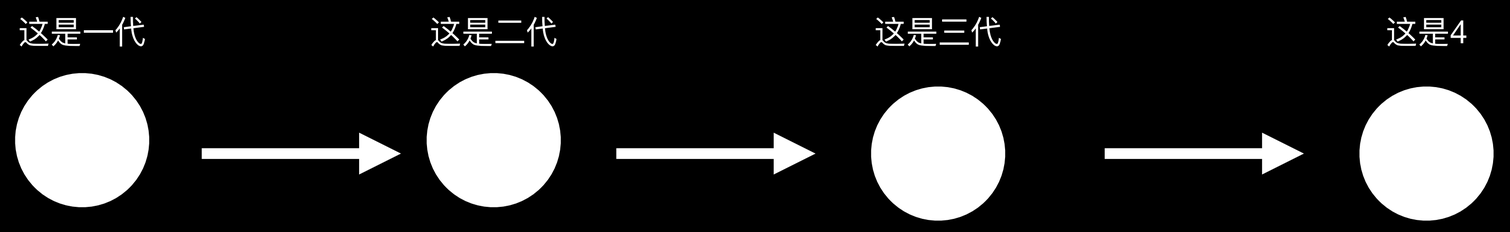
Thinking of history: story of snapshots

- // Skip the definition of snapshots now



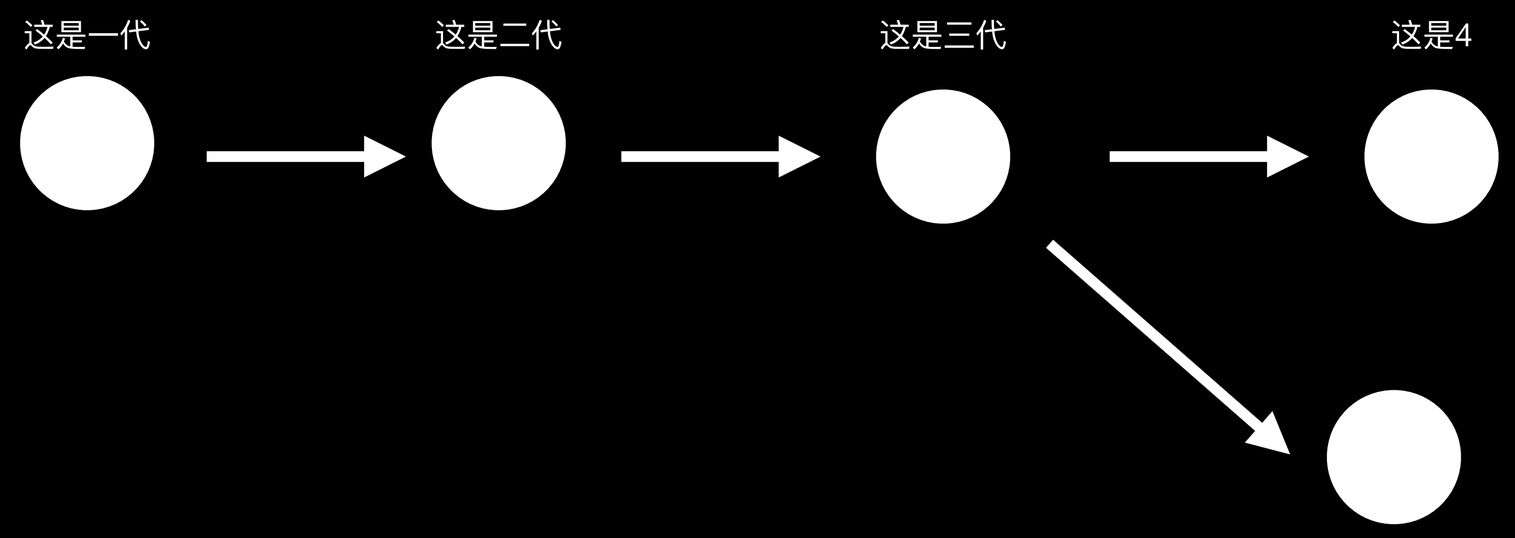
Thinking of history: story of snapshots

- // 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



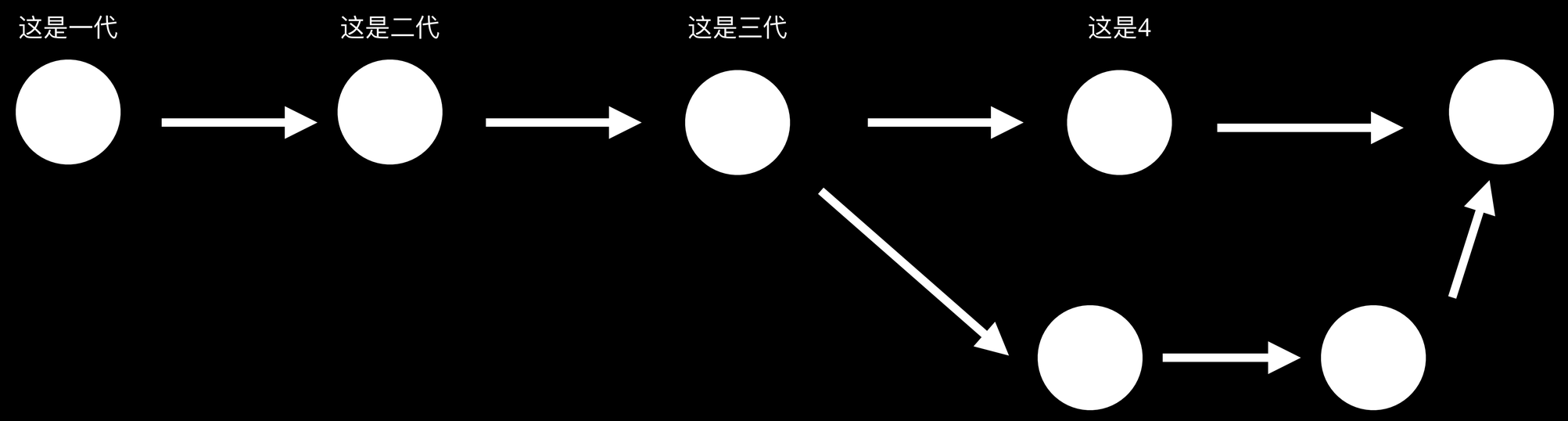
Thinking of history: story of snapshots

- // 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



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



Commit/Snapshot: who are you?

Commit/Snapshot: who are you?

- Snapshot is a collection of files and folders within some top-level directory
- File is called a "*blob*": a bunch of bytes.
- A directory is called a "*tree*": maps names to *blobs* or *trees*
  - directories can contain other directories

Commit/Snapshot: who are you?

- Snapshot is a collection of files and folders within some top-level directory
- File is called a "*blob*": a bunch of bytes.
- A directory is called a "*tree*": maps names to *blobs* or *trees*
  - directories can contain other directories

```
<root> (tree)
|
+- foo (tree)
|  |
|  + bar.txt (blob, contents = "hello world")
|
+- baz.txt (blob, contents = "git is wonderful")
```

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
}
```

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

SHA-1 is not for Human, References are

- Human-readable names for SHA-1 hashes, called *references*
  - *References are mutable*
  - *E.g., the master/main references usually point to the latest commit in the main branch of development*

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)
```

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".

Command (finally…

- 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*

Scenario-1: work on a local project

- Start a new project with *git init*
- Check status using *git status*

```
┌─dd@dd-PC7 ~/devlop/git-tutorial
└─$ ls
┌─dd@dd-PC7 ~/devlop/git-tutorial
└─$ git init
Initialized empty Git repository in /home/dd/devlop/git-tutorial/.git/
┌─dd@dd-PC7 ~/devlop/git-tutorial <main>
└─$ git status
On branch main

No commits yet

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

Thinking using data model (TUDM): Empty project, no commits, no history

Scenario-1: work on a local project

- Start a new project with *git init*
- Check status using *git status*

```
┌─dd@dd-PC7 ~/devlop/git-tutorial
└─$ ls
┌─dd@dd-PC7 ~/devlop/git-tutorial
└─$ git init
Initialized empty Git repository in /home/dd/devlop/git-tutorial/.git/
┌─dd@dd-PC7 ~/devlop/git-tutorial <main>
└─$ git status
On branch main

No commits yet

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

```
┌─dd@dd-PC7 ~/devlop/git-tutorial <main>
└─$ ls -alh
total 12K
drwxrwxr-x  3 dd dd 4.0K 11月  9 20:59 .
drwxrwxr-x 19 dd dd 4.0K 11月  9 20:59 ..
drwxrwxr-x  7 dd dd 4.0K 11月  9 20:59 .git
```

Thinking using data model (TUDM): Empty project, no commits, no history

Scenario-1: work on a local project

- Start a new project with *git init*
- Check status using *git status*

```
┌─dd@dd-PC7 ~/devlop/git-tutorial
└─$ ls
┌─dd@dd-PC7 ~/devlop/git-tutorial
└─$ git init
Initialized empty Git repository in /home/dd/devlop/git-tutorial/.git/
┌─dd@dd-PC7 ~/devlop/git-tutorial <main>
└─$ git status
On branch main

No commits yet

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

```
┌─dd@dd-PC7 ~/devlop/git-tutorial <main>
└─$ ls -alh
total 12K
drwxrwxr-x  3 dd dd 4.0K 11月  9 20:59 .
drwxrwxr-x 19 dd dd 4.0K 11月  9 20:59 ..
drwxrwxr-x  7 dd dd 4.0K 11月  9 20:59 .git
```

Git manages the project using .git/

Thinking using data model (TUDM): Empty project, no commits, no history

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-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)
```

```
┌─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
```

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

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)
```

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)
```

TUDM: A list of snapshots/commits. Head/main are referneces.

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
```

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
```

TUDM: We can turn to any prior snaptshot/commit using Git.

Scenario-1: work on a local project [5]

- Show changes on staging : *git checkout [commit_id]*

```
┌─dd@dd-PC7 ~/devlop/git-tutorial <main>
└─$ cat hello.txt
hello git
┌─dd@dd-PC7 ~/devlop/git-tutorial <main>
└─$ echo "new line" >> hello.txt
┌─dd@dd-PC7 ~/devlop/git-tutorial <main*>
└─$ cat hello.txt
hello git
new line
```

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

diff --git a/hello.txt b/hello.txt
index 8d0e412..b754b8d 100644
--- a/hello.txt
+++ b/hello.txt
@@ -1 +1,2 @@
 hello git
+new line
(END)
```

Scenario-1: work on a local project [5]

- Show changes on staging : *git checkout [commit_id]*

```
┌─dd@dd-PC7 ~/devlop/git-tutorial <main>
└─$ cat hello.txt
hello git
┌─dd@dd-PC7 ~/devlop/git-tutorial <main>
└─$ echo "new line" >> hello.txt
┌─dd@dd-PC7 ~/devlop/git-tutorial <main*>
└─$ cat hello.txt
hello git
new line
```

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

diff --git a/hello.txt b/hello.txt
index 8d0e412..b754b8d 100644
--- a/hello.txt
+++ b/hello.txt
@@ -1 +1,2 @@
 hello git
+new line
(END)
```

TUDM: We can turn to any prior snaptshot/commit using Git.

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
  - ...

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

- Formats on Linux community

Cases on RISC-V OpenSBI project

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

- Formats on Linux community

```
commit 723aa88ff4cc44230cf871bda3199051130032 79
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>
```

Cases on RISC-V OpenSBI project

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

- Formats on Linux community

```
commit 723aa88ff4cc44230cf871bda3199051130032 79
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

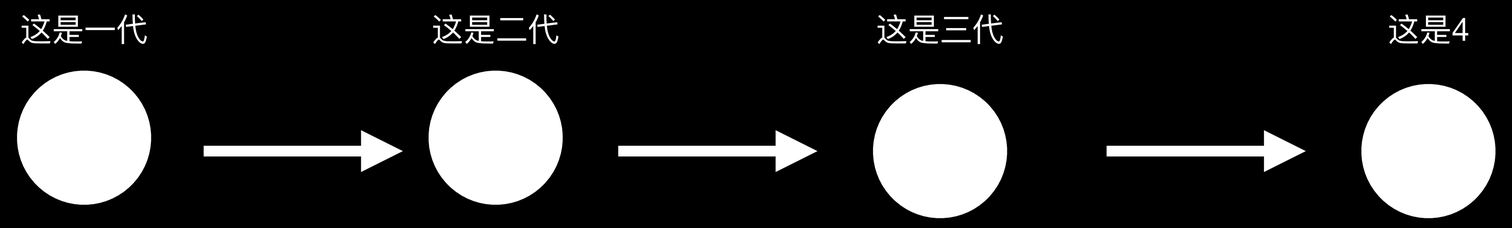
3. Your signed-off info, add “-s” during git commit

Cases on RISC-V OpenSBI project

Command (finally…[2]

- Branching and merging

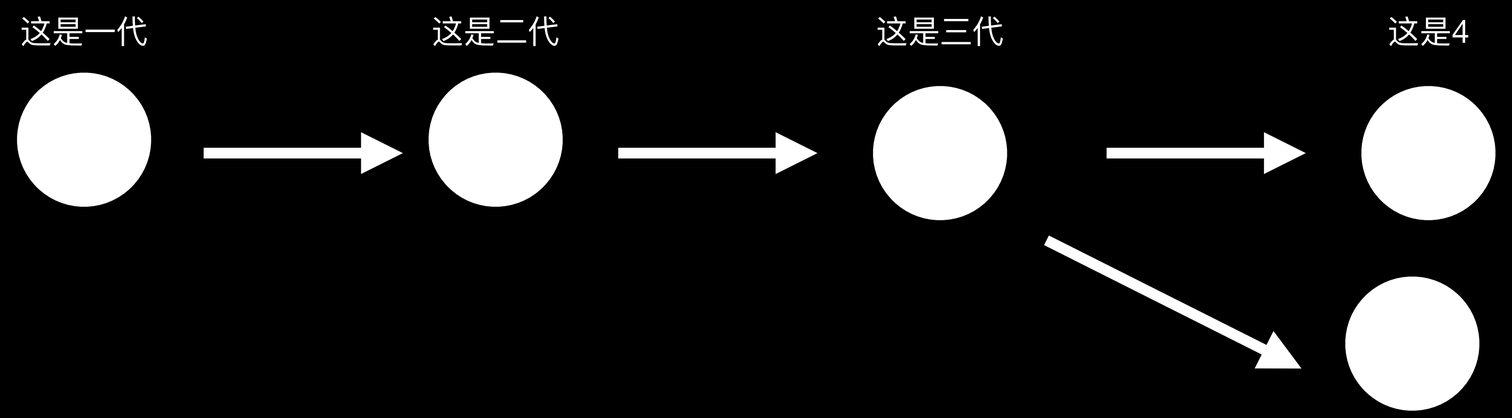
- *git branch: shows branches*
- *git branch <name>: creates a branch*
- *git checkout -b <name>: creates a branch and switches to it*
  - *same as git branch <name>; git checkout <name>*
- *git merge <revision>: merges into current branch*
- *git rebase: rebase set of patches onto a new base*



Command (finally…[2]

- Branching and merging

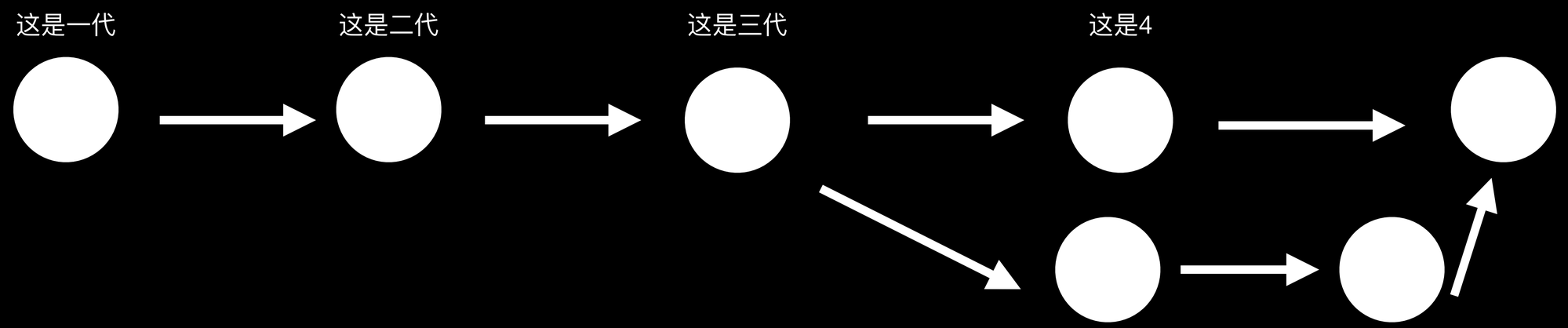
- *git branch: shows branches*
- *git branch <name>: creates a branch*
- *git checkout -b <name>: creates a branch and switches to it*
  - *same as git branch <name>; git checkout <name>*
- *git merge <revision>: merges into current branch*
- *git rebase: rebase set of patches onto a new base*



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*



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)
```

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)
```

TUDM: Create a new reference named debug (i.e., new branch)

Scenario-2: Debugging

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

Scenario-2: Debugging

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

```
dd@dd-PC7 ~/devlop/git-tutorial <debug>
$ cat world.txt
bye worlds
dd@dd-PC7 ~/devlop/git-tutorial <debug>
$ echo "this is debug info" >> world.txt
dd@dd-PC7 ~/devlop/git-tutorial <debug*>
$ cat world.txt
bye worlds
this is debug info
dd@dd-PC7 ~/devlop/git-tutorial <debug*>
$ git commit -asm "debug: add debug info"
[debug 86a9fe1] debug: add debug info
 1 file changed, 1 insertion(+)
```

Scenario-2: Debugging

- Merge debug branch into main: *gi*
-

```
dd@dd-PC7 ~/devlop/git-tutorial <debug>
$ cat world.txt
bye worlds
dd@dd-PC7 ~/devlop/git-tutorial <debug>
$ echo "this is debug info" >> world.txt
dd@dd-PC7 ~/devlop/git-tutorial <debug*>
$ cat world.txt
bye worlds
this is debug info
dd@dd-PC7 ~/devlop/git-tutorial <debug*>
$ git commit -asm "debug: add debug info"
[debug 86a9fe1] debug: add debug info
 1 file changed, 1 insertion(+)
```

```
commit 86a9fe14cc20f1061d9b803472ef604adc2e654c (HEAD -> debug)
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>

commit 78db867f3262427279c328f46f3ad5a96e936a02 (main)
Author: Dong Du <dd_nirvana@sjtu.edu.cn>
Date:   Tue Nov 9 21:33:48 2021 +0800

    update hello

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

commit ffe8f4238a08e7c03703bcb767d9afa5879937cf
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)
```

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(+)
```

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(+)
```

TUDM: Create a new commit using multiple parents

Scenario-2: Debugging

Scenario-2: Debugging

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

Scenario-2: Debugging

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

Scenario-2: Debugging

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

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*

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

Scenario-2-1: Debugging

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

Credits: cases from *git help rebase*

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
```

Credits: cases from *git help rebase*

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
```

Credits: cases from *git help rebase*

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

Credits: cases from *git help rebase*

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
```

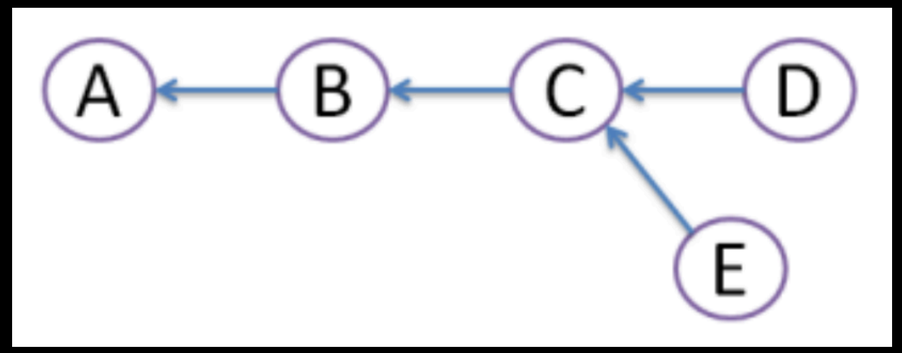
git rebase master topic

What's the differences between rebase and merge?

Credits: cases from *git help rebase*

Scenario-2-1: Debugging

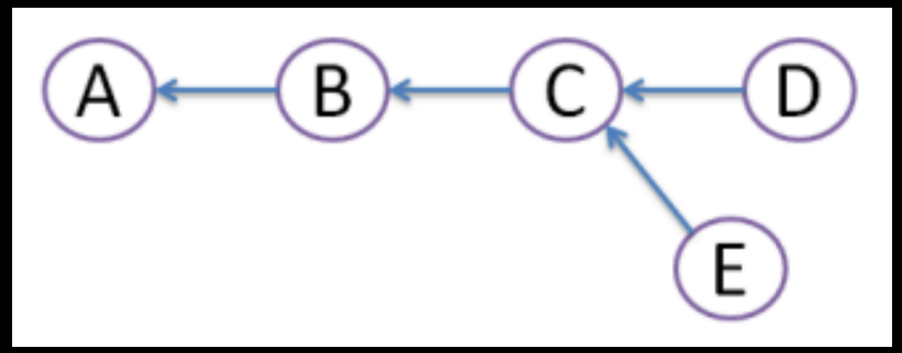
- Rebase vs. Merge



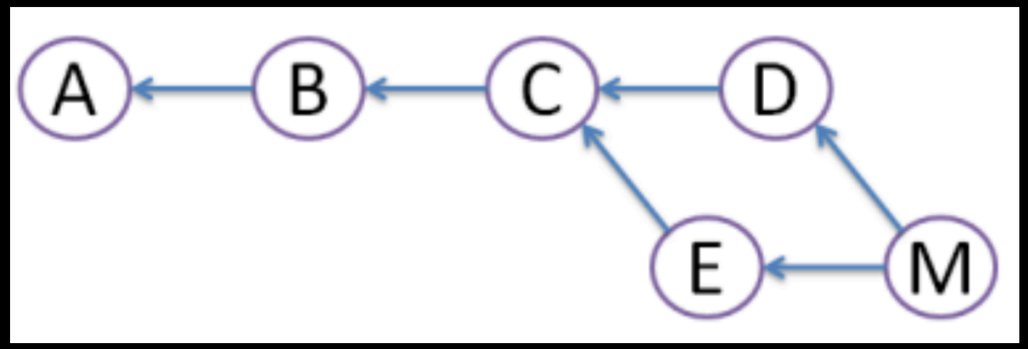
Credits: figs from stackoverflow (https://stackoverflow.com/questions/16666089/whats-the-difference-between-git-merge-and-git-rebase/25267150)

Scenario-2-1: Debugging

- Rebase vs. Merge



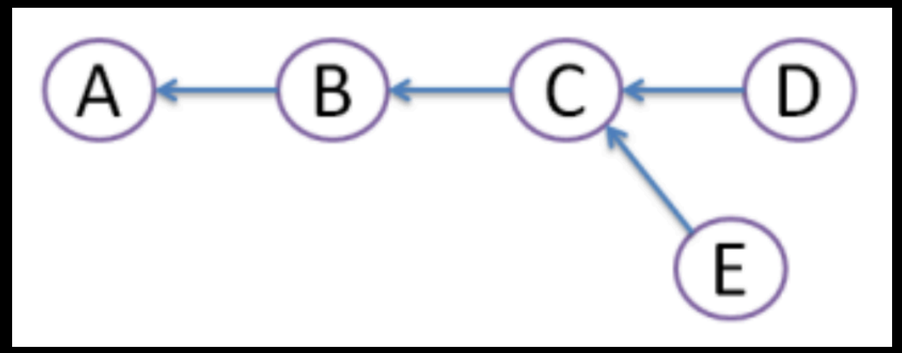
Git merge



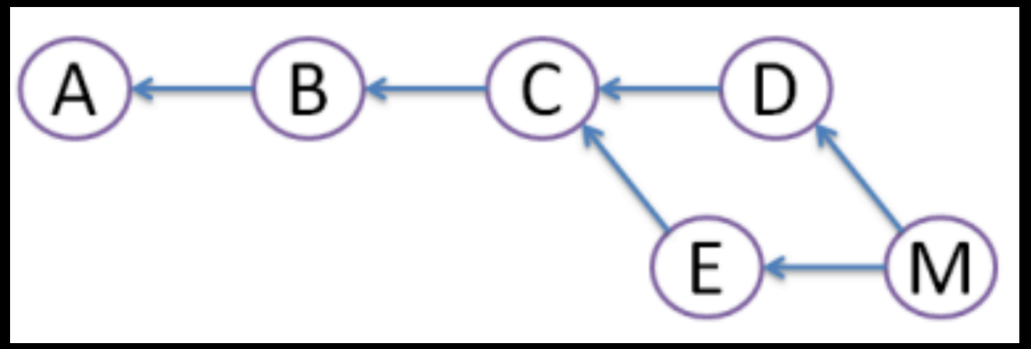
Credits: figs from stackoverflow (https://stackoverflow.com/questions/16666089/whats-the-difference-between-git-merge-and-git-rebase/25267150)

Scenario-2-1: Debugging

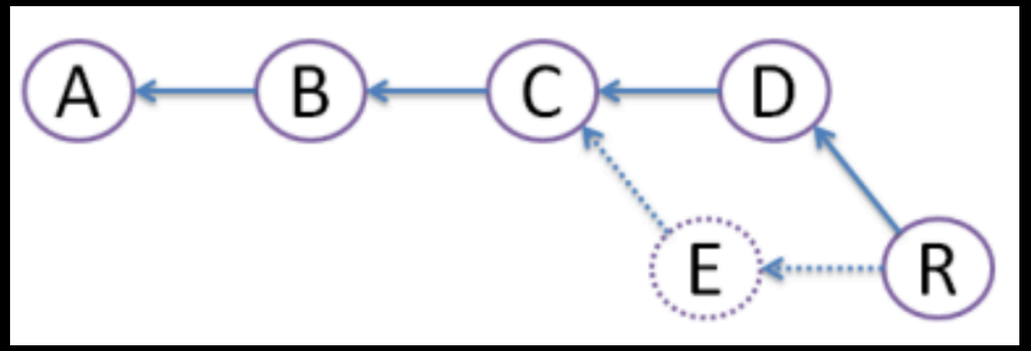
- Rebase vs. Merge



Git merge



Git rebase



Credits: figs from stackoverflow (https://stackoverflow.com/questions/16666089/whats-the-difference-between-git-merge-and-git-rebase/25267150)

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
```

Credits: cases from *git help rebase*

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

Credits: cases from *git help rebase*

Scenario-2-1: Debugging

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

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

?

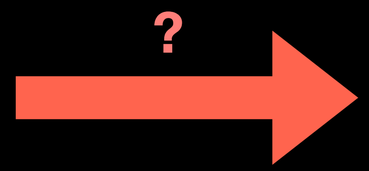
```
                    H'--I'--J'  topicB
                   /
                   | E---F---G  topicA
                   |/
A---B---C---D  master
```

Credits: cases from *git help rebase*

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

Credits: cases from *git help rebase*

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
```

Credits: cases from *git help rebase*

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

Credits: cases from *git help rebase*

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
```

Credits: cases from *git help rebase*

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

- 基于Git的代码托管平台
  - Github（网络不一定好）
  - Gitee（国内用还是很靠谱的）
  - Gitlab（实验室项目）

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
```

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*

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>
```

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
#
```

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>
```

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

Scenario-4: You will make mistakes, certainly

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

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 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 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
```

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 committe
  (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")
```

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 committe
  (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")
```

TUDM: Manage your staging area

Scenario-4: You will make mistakes, certainly

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

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
```

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

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*

Scenario-5: Git can do more for you

- Working in a team, who write the bug code?: *git blame*

Scenario-5: Git can do more for you

- Working in a team, who write the bug code?: *git blame*

```
dd@dd-PC7 ~/devlop/opensbi/opensbi <master*>
$ git blame README.md
```

Scenario-5: Git can do more for you

- Working in a team, who write th

```
dd@dd-PC7 ~/devlop/opensbi/opensbi <master*>
$ git blame README.md
```

```
79bfd67f (Atish Patra    2020-06-04 23:31:48 -0700   1) RISC-V Open Source Supervisor
Binary Interface (OpenSBI)
^9e8ff05 (Anup Patel     2018-12-11 19:24:06 +0530   2) ==============================
==========================
^9e8ff05 (Anup Patel     2018-12-11 19:24:06 +0530   3)
c2286b6f (Anup Patel     2020-05-06 12:16:48 +0530   4) Copyright and License
c2286b6f (Anup Patel     2020-05-06 12:16:48 +0530   5) --------------------
c2286b6f (Anup Patel     2020-05-06 12:16:48 +0530   6)
c2286b6f (Anup Patel     2020-05-06 12:16:48 +0530   7) The OpenSBI project is copyrig
ht (c) 2019 Western Digital Corporation
c2286b6f (Anup Patel     2020-05-06 12:16:48 +0530   8) or its affiliates and other co
ntributors.
c2286b6f (Anup Patel     2020-05-06 12:16:48 +0530   9)
c2286b6f (Anup Patel     2020-05-06 12:16:48 +0530  10) It is distributed under the te
rms of the BSD 2-clause license
c2286b6f (Anup Patel     2020-05-06 12:16:48 +0530  11) ("Simplified BSD License" or "
FreeBSD License", SPDX: *BSD-2-Clause*).
c2286b6f (Anup Patel     2020-05-06 12:16:48 +0530  12) A copy of this license with Op
enSBI copyright can be found in the file
c2286b6f (Anup Patel     2020-05-06 12:16:48 +0530  13) [COPYING.BSD].
c2286b6f (Anup Patel     2020-05-06 12:16:48 +0530  14)
c2286b6f (Anup Patel     2020-05-06 12:16:48 +0530  15) All source files in OpenSBI co
ntain the 2-Clause BSD license SPDX short
c2286b6f (Anup Patel     2020-05-06 12:16:48 +0530  16) identifier in place of the ful
l license text.
c2286b6f (Anup Patel     2020-05-06 12:16:48 +0530  17)
c2286b6f (Anup Patel     2020-05-06 12:16:48 +0530  18) ```
c2286b6f (Anup Patel     2020-05-06 12:16:48 +0530  19) SPDX-License-Identifier:    BS
D-2-Clause
c2286b6f (Anup Patel     2020-05-06 12:16:48 +0530  20) ```
c2286b6f (Anup Patel     2020-05-06 12:16:48 +0530  21)
c2286b6f (Anup Patel     2020-05-06 12:16:48 +0530  22) This enables machine processin
g of license information based on the SPDX
c2286b6f (Anup Patel     2020-05-06 12:16:48 +0530  23) License Identifiers that are a
vailable on the [SPDX] web site.
```

Scenario-5: Git can do more for you

- Working in a team, who write th

```
dd@dd-PC7 ~/devlop/opensbi/opensbi <master*>
$ git blame README.md
```

```
79bfd67f (Atish Patra    2020-06-04 23:31:48 -0700   1) RISC-V Open Source Supervisor
Binary Interface (OpenSBI)
^9e8ff05 (Anup Patel     2018-12-11 19:24:06 +0530   2) ==============================
==========================
^9e8ff05 (Anup Patel     2018-12-11 19:24:06 +0530   3)
c2286b6f (Anup Patel     2020-05-06 12:16:48 +0530   4) Copyright and License
c2286b6f (Anup Patel     2020-05-06 12:16:48 +0530   5) --------------------
c2286b6f (Anup Patel     2020-05-06 12:16:48 +0530   6)
c2286b6f (Anup Patel     2020-05-06 12:16:48 +0530   7) The OpenSBI project is copyrig
ht (c) 2019 Western Digital Corporation
c2286b6f (Anup Patel     2020-05-06 12:16:48 +0530   8) or its affiliates and other co
ntributors.
c2286b6f (Anup Patel     2020-05-06 12:16:48 +0530   9)
c2286b6f (Anup Patel     2020-05-06 12:16:48 +0530  10) It is distributed under the te
rms of the BSD 2-clause license
c2286b6f (Anup Patel     2020-05-06 12:16:48 +0530  11) ("Simplified BSD License" or "
FreeBSD License", SPDX: *BSD-2-Clause*).
c2286b6f (Anup Patel     2020-05-06 12:16:48 +0530  12) A copy of this license with Op
enSBI copyright can be found in the file
c2286b6f (Anup Patel     2020-05-06 12:16:48 +0530  13) [COPYING.BSD].
c2286b6f (Anup Patel     2020-05-06 12:16:48 +0530  14)
c2286b6f (Anup Patel     2020-05-06 12:16:48 +0530  15) All source files in OpenSBI co
ntain the 2-Clause BSD license SPDX short
c2286b6f (Anup Patel     2020-05-06 12:16:48 +0530  16) identifier in place of the ful
l license text.
c2286b6f (Anup Patel     2020-05-06 12:16:48 +0530  17)
c2286b6f (Anup Patel     2020-05-06 12:16:48 +0530  18) ```
c2286b6f (Anup Patel     2020-05-06 12:16:48 +0530  19) SPDX-License-Identifier:    BS
D-2-Clause
c2286b6f (Anup Patel     2020-05-06 12:16:48 +0530  20) ```
[illegible]                                          21)
[illegible]                                          22) This enables machine processin
[illegible]
c2286b6f (Anup Patel     2020-05-06 12:16:48 +0530  23) License Identifiers that are a
vailable on the [SPDX] web site.
```

TUDM: The latest history of each line: commit-id/authors/..

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
```

Summary and Q&A?

- *Basic knowledge about git is necessary*
- *More “advanced” tools (e.g., vscode) may help you use Git*
- *Try to read Pro-Git (https://git-scm.com/book/en/v2) if you want to know more*
- *Thx*