

??

- [Git 入門 \(Basic\)](#)
- [Git 上級 \(Advanced\)](#)
- [commit する .git がある](#)
- [Git 初期化 する .git がある \(1\)](#)
- [Git 初期化 する .git がある \(2\)](#)
- [README](#)
- [LICENSE](#)

Git 是什么 (Basic)

1. Git 是什么？为什么需要它？

Git 是一个分布式版本控制系统 (Distributed Version Control System, DVCS)。

“版本控制” = 记录文件的变化 (如, 代码 的 修改)

2. Git 的工作流程

本地工作目录 (Working Directory)	→	暂存区 / 索引 (Staging/Index)	→	本地仓库 (Repository/.git)
修改文件		git add		git commit

本地工作目录	本地仓库
存放正在开发的文件	存放提交后的文件 / 历史记录
通过 git add 将文件放入暂存区 (Index)	通过 git commit 提交文件到仓库 (git add)
通过 git checkout 从仓库恢复文件	通过 git pull 从远程仓库更新本地仓库

3. Git 的底层结构

Git 的底层结构由 4 个主要部分组成：

对象类型	描述	示例
Blob	存储文件内容 (binary large object)	hello.java 文件
Tree	存储目录结构 (目录和文件的集合)	项目根目录
Commit	记录当前提交的状态 + 指向父提交 + tree	"提交信息"
Tag	标记特定的提交点	v1.0.0

tree 对象 , tree 的 blob 对象 .

```
[commit]
|
v
[tree]
├─ [blob] hello.txt
└─ [blob] app.js
```

4. HEAD??

HEAD → 指向 哪个 commit 或 分支

- HEAD → main
- `git checkout` 分支 HEAD 指向 分支
- `git reset` **HEAD** 指向 指定 commit

5. ?????

分支 指向 哪个 commit 或 分支 .

```
A---B---C ← main (分支)
      ↑
    HEAD
```

- 分支 → 分支
- 分支 → 分支 分支 分支 分支 分支

6. Reflog??

`git reflog`

→ HEAD 指向 哪个 commit 或 分支

- `reset` `checkout` 分支 分支
- 分支 Git HEAD 指向 分支 分支

7. .git ????? ?? ??

.git/

└─ objects/

└─ refs/

└─ HEAD

└─ index

└─ logs/

└─ config

← 00, 00, 00, 00 00 000

← 0000, 00 00 000

← 00 0000 000

← 0000 00

← reflog 00 00

← Git 00

? ??

??	??
Commit	00 000 + 00 00 00
Blob	00 00 00
Tree	00 00
Branch	000 0000 000
HEAD	00 000 or 00
Staging Area	00 000 0000 00
Reflog	HEAD0 00 00 00 00 (000 00)

Git 对象 (Advanced)

1. Git 对象 SHA-1 是什么

1.1 是什么

Git 对象 (blob, tree, commit 等) 都是 **SHA-1** 哈希值。每个对象都有一个唯一的 SHA-1 哈希值。

```
“对象” 大小 40 字节 16 字节 对象
```

1.2 是什么

```
e83c5163... ← 对象, 对象, 对象 对象 对象 对象
```

1.3 是什么 是什么 是什么

1. Git 对象 对象 + 对象 + 对象 对象 对象
对象 : blob 12\0Hello World\n
2. 对象 SHA-1 对象 对象 对象 对象

```
echo "Hello World" | git hash-object --stdin  
→ 557db03de997c86a4a028e1ebd3a1ceb225be238
```

2. Git 打包文件 是什么 是什么

2.1 是什么

Git 对象 对象 对象 .git/objects 对象 对象 , 对象 对象 对象 对象 .
对象 对象 对象 **packfile(.pack)** 对象 .

2.2 packfile???

“Git 的 packfile 是啥？”

2.3 ?? ??

```
.git/
├─ objects/
│   └─ pack/
│       ├── pack-xxxxx.pack ← 打包后的文件
│       └─ pack-xxxxx.idx ← 索引文件
```

2.4 ?? ???

```
git gc # 打包 packfile (garbage collection)
```

2.5 ??

操作	描述
clone	克隆仓库，包括所有对象和索引文件
fetch, push	克隆、获取、推送操作
packfile	打包文件，将对象打包成 packfile 和索引文件

3. Git vs SVN ??

操作	Git	SVN
clone	克隆仓库 (DVCS)	克隆仓库 (CVCS)
fetch, push	获取、推送操作	获取、推送操作
packfile	打包文件 (packfile)	打包文件 (packfile)
merge & rebase	X (不支持)	O (支持)
	不支持 (不支持)	支持 (不支持)
	不支持	支持

3.1 分布式版本控制

| Git 是一个分布式版本控制系统，与集中式版本控制系统（如 SVN）相比，Git 是 lightweight 的。
→ 每个开发者都有一个完整的版本库副本，可以离线工作，无需连接到中央服务器。

| SVN 是一个集中式版本控制系统，所有文件都存储在中央服务器上，客户端需要连接到服务器才能进行版本控制。
→ 客户端只能看到最新版本，无法看到历史版本，且需要连接到服务器才能进行版本控制。

Git 与 SVN 的区别

特性	Git
SHA-1 哈希	Git 使用 SHA-1 哈希来唯一标识每个文件版本，确保文件的完整性和一致性。
packfile	Git 使用 packfile 来存储文件内容，可以高效地存储和传输大量文件。
Git vs SVN	Git 是分布式版本控制系统，而 SVN 是集中式版本控制系统。Git 支持离线工作，而 SVN 需要连接到服务器。

commit ??? .git ?? ??

????

- 1. hello.txt
- 2. git add hello.txt
- 3. git commit -m "Add hello"

1. ?? (Working Directory)

```
echo "Hello Git" > hello.txt
```

-
- Git

2. git add hello.txt

```
git add hello.txt
```

?? ??:

.git/index	(, ,)
.git/objects/	hello.txt blob (SHA-1)

?? :

```
.git/  
└─ objects/
```


└─ 3a/

└─ 3efbed4aef... ← hello.txt blob

3. `git commit -m "Add hello"`

?? ??:

`.git/objects/`

对象	内容
tree 47	目录 (hello.txt 指向 3efbed4aef...)
commit a1	元数据, 指向父提交, 指向 tree 对象

```
.git/objects/  
├─ 3a/... (blob)  
├─ 47/... (tree)  
└─ a1/... (commit)
```

`.git/refs/heads/main`

- 指向 47 的树对象

`refs/heads/main → a1b2c3d4e5...`

`.git/HEAD`

- 指向 `ref: refs/heads/main` 的指针 → main 分支

`.git/logs/HEAD & logs/refs/heads/main`

- HEAD 的日志 (reflog)

`0000000 → a1b2c3d4 Commit: Add hello`

?? ?? (?? 1? ?)

```

Working Directory      ← hello.txt
  ↓ git add
.git/index             ← 100644 00 00
  ↓ git commit
.git/objects/
  ├── blob (100 00)
  ├── tree (100644 00)
  └── commit (100644)

.git/refs/heads/main   ← 100644 → 00 00
  ↑
.git/HEAD              ← 100 100644 00
.git/logs/             ← HEAD/100644 00 00

```

?? ?? (?? ?)

```

cat .git/HEAD
# ref: refs/heads/main

cat .git/refs/heads/main
# a1b2c3d4e5...

git ls-tree HEAD
# 100644 blob 3a3efbed...hello.txt

git cat-file -p <100644>
# tree 47fa...
# author ...
# message: Add hello

```

??

??	??
git add	blob ?? ?? + index ??
git commit	tree + commit ?? ?? , ??? ?? ??

`.git` ☐ ☐ objects, refs, HEAD, logs ☐ ☐

objects, refs, HEAD, logs  

Git ????? ?? .git ?? ?? ??(1?)

1. git merge ? ?? ??

1.1 ?? ???

```
git checkout main
git merge feature
```

1.2 .git ?? ??

?? ??	?? ??
.git/objects/	?? ?? ?? merge commit ?? ??
.git/refs/heads/main	??? ???? merge ???? ??
.git/HEAD	??? ref: refs/heads/main ??
.git/MERGE_HEAD	?? ?? ?? (feature) ?? ?? (?? ?? ? ???)
.git/logs/HEAD	HEAD ?? ?? ?? (reflog)

1.3 ?? ??

```
A---B---C ← main
      \
      D---E ← feature

► git merge feature

?:

A---B---C-----M (merge commit) ← main, HEAD
```

\ /
D---E

2. git rebase ? ? ? ?

2.1 ? ? ? ? ?

```
git checkout feature  
git rebase main
```

2.2 .git ? ? ? ?

本地仓库	远程仓库
<code>.git/objects/</code>	<code>feature</code> <code>00</code>
<code>.git/refs/heads/feature</code>	<code>00</code>
<code>.git/REBASE_HEAD</code>	<code>00</code>
<code>.git/HEAD</code>	<code>ref: refs/heads/feature</code>
<code>.git/logs/HEAD</code>	<code>HEAD 00</code>

2.3 ? ? ? ?

```
main:    A---B---C  
feature:          D---E  
  
► git rebase main  
  
本地:  
main:    A---B---C  
feature:          D'---E' ← HEAD
```

3. `git reset` ? ?? ??

3.1 ?? ???

```
git reset --hard HEAD~1
```

3.2 `.git` ?? ??

□□ □□	□□ □□
<code>.git/refs/heads/main</code>	□□□ □□□ □□ □□□ □□
<code>.git/HEAD</code>	□□□ <code>ref: refs/heads/main</code>
<code>.git/index</code>	□□□ □□ □□□ □□□ □□ □□□
□□ □□□	<code>--hard</code> □□ □□ □□□
<code>.git/ORIG_HEAD</code>	□□ HEAD □□ □□□ □□
<code>.git/logs/HEAD</code>	□□ □□ □□ □□□

3.3 ?? ??

```
A---B---C ← main, HEAD
```

```
↑
```

```
ORIG_HEAD
```

```
► git reset --hard B
```

```
□□:
```

```
A---B ← main, HEAD
```

4. ?? HEAD ?? ??

?? ??	?? ??	??
.git/MERGE_HEAD	git merge ??	?? ?? ?? ??
.git/REBASE_HEAD	git rebase ??	?? ?? ??
.git/ORIG_HEAD	git reset, merge	?? ?? ??
.git/FETCH_HEAD	git fetch	???? ?? ??
.git/CHERRY_PICK_HEAD	git cherry-pick	?? ?? ??

5. ?? ???

??	?? ??	?? ??	HEAD ??	???? ??	?? ?? ??
merge				?? ?? ??	MERGE_HEAD
rebase	(rewrite)			?? ?? ??	REBASE_HEAD
reset --hard	(?? ????)			✓ ?? ??	ORIG_HEAD

```
git stash
```

파일명	내용
<code>.git/objects/</code>	객체 (blob, tree, commit, tag)의 해시 (40자리)로 인덱싱된 데이터 (2~3MB)
<code>.git/logs/refs/stash</code>	stash의 변경 내역 (commit, untracked files, index, working tree)
<code>.git/refs/stash</code>	stash의 현재 상태 (commit, untracked files, index, working tree)

```
.git/
├─ objects/      ← stash 00 00 00 00 00
├─ refs/stash    ← 00 stash 00
├─ logs/refs/stash ← stash 00 00
```

```
main:    A---B---C
          ↑
        HEAD / index
```

▶ git stash

```
##:
- ##  ## ##
- .git/refs/stash → stash ## ##
```

11

- `git stash` `git stash`
- `.git/refs/stash` → `stash` `git stash`

5. `git cherry-pick` ? ?? ??

5.1 ?? ???

```
git cherry-pick alb2c3d
```

5.2 `.git` ?? ??

?? ??	?? ??
<code>.git/objects/</code>	cherry-pick 对象 对象 对象 对象 对象
<code>.git/CHERRY_PICK_HEAD</code>	对象 对象 对象 对象 (对象 对象 对象 对象)
<code>.git/HEAD</code>	对象 对象 对象 对象
<code>.git/index</code>	对象 对象 对象

5.3 ?? ??

```
main:    A---B---C ← HEAD
feature:      D---E ← cherry-pick 对象
```

► `git cherry-pick E`

```
对象:
main:    A---B---C---E' ← HEAD
```

“E 对象 对象 对象 对象 E' 对象 对象

6. `git revert` ? ?? ??

6.1 ?? ???

```
git revert C
```

6.2 .git ?? ??

?? ??	?? ??
<code>.git/objects/</code>	revert ?? ?? ?? ??
<code>.git/index</code>	C ?? ?? ?? ?? ??
<code>.git/HEAD</code>	?? ?? ?? ??

```
“ revert ?? ?? ?? ?? ?? , "?? ?? ?? ?? " ?? ?? ?? ?? .
```

6.3 ?? ??

A---B---C ← HEAD

► git revert C

??:

A---B---C---C' ← HEAD (C ?? ?? ?? ??)

7. ?? ?? ?? (2?)

?? ??	?? ??	??
<code>.git/CHERRY_PICK_HEAD</code>	<code>git cherry-pick</code>	?? ?? ?? ?? ?? (?? ? ??)
<code>.git/REVERT_HEAD</code>	<code>git revert</code> (?? ?)	revert ?? ?? ?? ??
<code>.git/refs/stash</code>	<code>git stash</code>	?? ?? stash ??
<code>.git/logs/refs/stash</code>	<code>git stash</code>	stash ?? ?? ??

8. ?? ??? (2?)

명령어	대상 레퍼런스	대상 레퍼런스	HEAD 레퍼런스	대상 레퍼런스	대상 레퍼런스
stash	O (대상 레퍼런스)	대상 레퍼런스	대상 레퍼런스	refs/stash	대상 레퍼런스
cherry-pick	O	대상 레퍼런스	O	CHERRY_PICK_HEAD	대상 레퍼런스
revert	O	대상 레퍼런스	O	REVERT_HEAD	대상 레퍼런스

???

1. ?? ?? ??

```
.git/
├ HEAD                ← 当前 HEAD 指针 (即: ref: refs/heads/main)
├ config              ← Git 配置文件
├ description         ← 仓库描述文件
├ index               ← 本地索引文件 (包含已提交但未推送的更改)
├ objects/            ← Git 对象存储 (blob, tree, commit, tag)
│   └ xx/xxxx...      ← SHA-1 哈希值 对象 名称
│   └─ pack/          ← 打包后的对象 (.pack, .idx)
├ refs/              ← 分支、标签等引用
│   └ heads/          ← 本地分支 (即: main, feature)
│   └ remotes/        ← 远程分支 (即: origin/main)
│   └─ tags/          ← 标签
├ logs/              ← HEAD 和 refs 的日志 (reflog 记录)
│   └ HEAD            ← HEAD 的日志
│   └─ refs/          ← refs 的日志
├ info/              ← exclude 文件等
└─ hooks/            ← 钩子, 即 Git 触发的事件
```

2. ? ?? ?? ??

? `.git/HEAD`

- 本地 Git 仓库的 HEAD 指针
- 指向 `ref: refs/heads/main` 分支
- **HEAD** → 指向 → 分支

? `.git/objects/`

- Git 对象 : 存储 / 文件 / 目录 **SHA-1** 哈希值 名称

- `git` `add` `commit` `push`

```
.git/objects/
├─ a7/...
├─ d1/...
└─ pack/ ← packfile 100 1000
```

? `.git/refs/`

- `git` `refs` `heads` `main` `branch` `name`

```
.git/refs/heads/main → a1b2c3d4e5 (100 100)
```

? `.git/logs/`

- `git reflog` `HEAD` `branches` `remotes` `branch` `name`
- `HEAD`, `branches`, `remotes` `branch` `name` `name`

? `.git/index`

- `git` `add` `(index)` `branch` `name` `branch` `name` `branch` `name`
- `git add` → `index` `branch` → `git commit` `branch` `name`

? `.git/hooks/`

- `git` `hooks` `pre-commit` `post-commit` `pre-push` `branch` `name`

```
pre-commit, post-commit, pre-push
```

3. ?? ??

```
Working Directory
↓ git add
.git/index (Staging Area)
↓ git commit
.git/objects/ (100, 100, 100 100)
↓ 100 100 100
```

00.git/refs/heads/main

↑ HEAD → 00 000 00

??

0000	00
HEAD	00 000 00 000 000
refs	000 /00 → 00 00
objects	00 000 (SHA-1 00) 000
index	0000 00 00
logs	HEAD 0 000 00 00 (reflog)
hooks	Git 000 00 0 0000 0000

?????

Git 目录结构

```
├─ 1. Working Directory
|   └─ 存放开发中的文件 (即代码, 文件)
|
├─ 2. Index (Staging Area) ← .git/index
|   └─ git add 文件
|
├─ 3. Local Repository (.git/)
|   └─ 3.1 objects/
|       └─ blob: 文件内容
|           └─ tree: 目录结构
|               └─ commit: 提交信息 + 父提交哈希 + 时间戳
|
|   └─ 3.2 refs/
|       └─ heads/: 分支指针 (如 main, develop)
|           └─ remotes/: 远程分支指针
|               └─ tags/: 标签指针
|
|   └─ 3.3 HEAD
|       └─ 指向当前分支的指针 (如: ref: refs/heads/main)
|           └─ detached HEAD 状态
|
|   └─ 3.4 logs/
|       └─ HEAD 的提交历史 (reflog)
|
|   └─ 3.5 config
|       └─ 本地配置信息
|
|   └─ 3.6 hooks/
|       └─ pre-commit, post-commit 等钩子脚本
|
|   └─ 3.7 其他 HEAD 指针
|       └─ MERGE_HEAD      ← merge 操作时
|       └─ REBASE_HEAD    ← rebase 操作时
|       └─ ORIG_HEAD       ← reset 操作时
```

- └─ CHERRY_PICK_HEAD ← cherry-pick 时 指向 父分支
- └─ REVERT_HEAD ← revert 时 指向 父分支
- └─ FETCH_HEAD ← fetch 时 指向 远程分支

4. 常用命令

- └─ git add
 - └─ 将文件添加到 index 中
- └─ git commit
 - └─ index 中的文件写入 commit 中
 - └─ 生成 commit 消息
- └─ git reset
 - └─ HEAD/当前分支 指向 指定 commit
 - └─ --hard 将 index + 工作区 重置为指定 commit 的状态
- └─ git merge
 - └─ merge commit 生成
 - └─ MERGE_HEAD 指向 合并中的分支
- └─ git rebase
 - └─ 重写历史 (rewrite)
 - └─ REBASE_HEAD 指向 正在 rebasing 的分支
- └─ git cherry-pick
 - └─ 将指定 commit 合并到当前分支
 - └─ CHERRY_PICK_HEAD 指向 正在 cherry-pick 的 commit
- └─ git revert
 - └─ 将指定 commit 逆向
 - └─ REVERT_HEAD 指向 正在 revert 的 commit
- └─ git stash
 - └─ 将当前分支的工作区内容暂存
 - └─ refs/stash 指向 暂存的内容
 - └─ logs/refs/stash 记录暂存的内容

5. 其他命令

- └─ git reflog → HEAD, 记录 HEAD 的变动
- └─ git log → 查看历史
- └─ git cat-file -p <commit> → 查看 commit 内容
- └─ git fsck → 检查仓库完整性

??

对象	描述
blob	文件内容
tree	目录结构
commit	提交记录
HEAD	当前分支的指针
refs	分支、标签、远程仓库的引用
logs	HEAD/refs 的变更记录 (reflog)
index	下次提交要用的暂存区
stash	暂存未提交的更改 (暂存区)