|  |
| --- |
| **速查目录（建议做成册子的目录页）**   1. 初始化与配置：git init / git clone / git config 2. 状态与查看：git status / git diff / git log / git show / git blame / git grep 3. 添加与提交：git add / git commit / git restore --staged 4. 分支与切换：git branch / git switch（或 checkout）/ git merge / git rebase / git worktree 5. 远程协作：git remote / git fetch / git pull --rebase / git push 6. 标签与发布：git tag / git describe 7. 临时存货：git stash 8. 撤销与回滚：git restore / git reset / git revert / git checkout -- <file> / git reflog（救命绳） 9. 精准移植：git cherry-pick 10. 清理与维护：git clean / git rm / git mv / git gc 11. 历史改造：git rebase -i / git commit --amend / git format-patch / git apply / git am 12. 子模块与稀疏检出：git submodule / git sparse-checkout / （可选）git subtree 13. 调试定位：git bisect 14. 常用别名与钩子：git config alias.\* / .git/hooks/\* 15. .gitignore/属性：.gitignore / .gitattributes   下面按“是什么→什么时候用→怎么用→例子→注意坑”给到。能复制的命令全部给你放上。 |

目录

[**1) 初始化与配置** 2](#_Toc206932572)

[**git init** 2](#_Toc206932573)

[**git clone** 2](#_Toc206932574)

[**git config** 2](#_Toc206932575)

[**2) 状态与查看** 2](#_Toc206932576)

[**git status** 2](#_Toc206932577)

[**git diff** 2](#_Toc206932578)

[**git log** 3](#_Toc206932579)

[**git show** 3](#_Toc206932580)

[**git blame** 3](#_Toc206932581)

[**git grep** 3](#_Toc206932582)

[**3) 添加与提交** 3](#_Toc206932583)

[**git add** 3](#_Toc206932584)

[**git commit** 4](#_Toc206932585)

[**git restore --staged & git restore** 4](#_Toc206932586)

[**4) 分支与切换** 4](#_Toc206932587)

[**git branch** 4](#_Toc206932588)

[**git switch（更现代） / git checkout（传统）** 4](#_Toc206932589)

[**git merge** 4](#_Toc206932590)

[**git rebase** 5](#_Toc206932591)

[**git worktree** 5](#_Toc206932592)

[**5) 远程协作** 5](#_Toc206932593)

[**git remote** 5](#_Toc206932594)

[**git fetch / git pull --rebase / git push** 5](#_Toc206932595)

[**6) 标签与发布** 6](#_Toc206932596)

[**git tag / git describe** 6](#_Toc206932597)

[**7) 临时存货：git stash** 6](#_Toc206932598)

[**8) 撤销与回滚（从安全到危险）** 6](#_Toc206932599)

[**A. 最安全：git revert** 6](#_Toc206932600)

[**B. 中等风险：git restore（撤工作区/暂存）** 6](#_Toc206932601)

[**C. 危险但常用：git reset** 6](#_Toc206932602)

[**D. 绝望时的救命绳：git reflog** 7](#_Toc206932603)

[**9) 精准移植：git cherry-pick** 7](#_Toc206932604)

[**10) 清理与维护** 7](#_Toc206932605)

[**git clean** 7](#_Toc206932606)

[**git rm / git mv** 7](#_Toc206932607)

[**git gc** 7](#_Toc206932608)

[**11) 历史改造（本地私有分支用）** 8](#_Toc206932609)

[**交互式变基：git rebase -i** 8](#_Toc206932610)

[**修补信息：git commit --amend** 8](#_Toc206932611)

[**生成补丁 / 应用补丁：git format-patch / git apply / git am** 8](#_Toc206932612)

[**12) 子模块与稀疏检出** 8](#_Toc206932613)

[**git submodule** 8](#_Toc206932614)

[**git sparse-checkout** 8](#_Toc206932615)

[**13) 调试定位：git bisect** 9](#_Toc206932616)

[**14) 常用别名（懒人必配 🧰）** 9](#_Toc206932617)

[**15) .gitignore & .gitattributes（最常用片段）** 9](#_Toc206932618)

[**场景化工作流（拿去即用）** 10](#_Toc206932619)

[**A) 新建项目并首推** 10](#_Toc206932620)

[**B) 日常开发（保持历史干净）** 10](#_Toc206932621)

[**C) 代码评审前整理提交** 11](#_Toc206932622)

[**D) 修复线上 bug（hotfix）** 11](#_Toc206932623)

[**E) 临时切出审查别的分支，但不想提交未完成工作** 11](#_Toc206932624)

[**F) 提交写错了 / 不该包含某文件** 11](#_Toc206932625)

[**G) 合并冲突处理套路（无脑模板）** 11](#_Toc206932626)

[**H) 找回“误删分支/误 reset”的历史** 12](#_Toc206932627)

[**I) 只拉巨仓库的一角（稀疏检出）** 12](#_Toc206932628)

[**J) 从 A 分支“挑”一个修复到 B 分支** 12](#_Toc206932629)

[**常见报错速译 & 解法 🩹** 12](#_Toc206932630)

[**团队建议（不学理论也能不踩坑）🧭** 13](#_Toc206932631)

[**进阶但超有用的几味小料 🍜** 13](#_Toc206932632)

**1) 初始化与配置**

**git init**

* **作用**：在当前文件夹创建一个新的 Git 仓库。
* **场景**：新项目、老项目要开始版本管理。
* **用法**：
* git init
* **注意**：默认分支名可能是 master 或 main，统一成 main：
* git branch -M main

**git clone**

* **作用**：从远程复制一个仓库到本地。
* **场景**：加入团队、拉现成项目。
* **用法**：
* git clone <repo-url>
* git clone -b develop <repo-url> # 直接克隆指定分支

**git config**

* **作用**：设置用户信息、别名、颜色、工具。
* **场景**：第一次用 Git、个性化设置。
* **常用**：
* git config --global user.name "你的名字"
* git config --global user.email "你的邮箱"
* git config --global init.defaultBranch main
* git config --global pull.rebase true # 推荐：pull 默认 rebase
* git config --global core.autocrlf input # mac/linux 常用；Windows 可用 true
* git config --global color.ui auto

**2) 状态与查看**

**git status**

* **作用**：看“我现在有什么改动”。
* **场景**：提交前/冲突后。
* **用法**：git status

**git diff**

* **作用**：查看代码差异。
* **场景**：提交前复查、冲突分析。
* **用法**：
* git diff # 工作区 vs 暂存区
* git diff --staged # 暂存区 vs 最新提交
* git diff branchA..branchB

**git log**

* **作用**：查看提交历史。
* **常用花样**：
* git log --oneline --graph --decorate --all
* git log --since="2025-08-01" --author="Alice" -- path/to/file
* git log -p -1 # 最近一次提交的补丁

**git show**

* **作用**：查看某个提交/标签/对象的详情。
* **用法**：
* git show HEAD
* git show <commit-id> -- path/to/file

**git blame**

* **作用**：是谁在什么时候改了这行。
* **用法**：git blame path/to/file

**git grep**

* **作用**：在版本库内全文搜索。
* **用法**：
* git grep "正则或文本"
* git grep -n "TODO" -- '\*.js'

**3) 添加与提交**

**git add**

* **作用**：把改动加入“暂存区”。
* **常用**：
* git add -A # 添加所有变化（新增/修改/删除）
* git add . # 添加当前目录所有变化
* git add -p # 交互式分块添加（精细！）

**git commit**

* **作用**：把暂存区变成一次提交。
* **常用**：
* git commit -m "feat: 用户可导出报表"
* git commit --amend # 改写上一条提交（别在公共分支上这么干）
* **信息建议**：<type>: <subject>（feat/fix/docs/refactor/chore/test）

**git restore --staged & git restore**

* **作用**：撤销暂存/工作区改动（不删历史）。
* **用法**：
* git restore --staged path/file # 从暂存区撤回
* git restore path/file # 丢弃工作区改动（谨慎）

**4) 分支与切换**

**git branch**

* **作用**：查看/新建/删除分支。
* **用法**：
* git branch
* git branch -a # 包括远程
* git branch -d feature/x # 删除已合并的本地分支
* git branch -D feature/x # 强制删除

**git switch（更现代） / git checkout（传统）**

* **作用**：切换分支、基于当前提交新建分支。
* **用法**：
* git switch -c feature/login # 新建并切换
* git switch main # 切换
* # 旧：git checkout -b feature/login

**git merge**

* **作用**：把一个分支合并到当前分支（生成合并提交）。
* **用法**：
* git switch main
* git merge feature/login # 可能触发冲突，解决后：
* git add <冲突文件>
* git commit # 完成合并

**git rebase**

* **作用**：把分支基于目标分支“变基”，历史更直。
* **推荐**：**私人分支**上用；**公共分支**慎用。
* **用法**：
* git switch feature/login
* git fetch origin
* git rebase origin/main
* # 解决冲突 -> git add . -> git rebase --continue

**git worktree**

* **作用**：同一仓库开多工作目录，多个分支并行开发，省去反复切换。
* **用法**：
* git worktree add ../proj-fix hotfix/urgent
* git worktree list
* git worktree remove ../proj-fix

**5) 远程协作**

**git remote**

* **作用**：管理远程源。
* **用法**：
* git remote -v
* git remote add origin <repo-url>
* git remote set-url origin <new-url>

**git fetch / git pull --rebase / git push**

* **fetch**：只拉元数据与对象，不合并。
* git fetch origin
* **pull --rebase**：拉最新并把本地提交“平铺”到最新之上（历史更干净）。
* git pull --rebase
* **push**：推送本地分支/标签到远程。
* git push origin HEAD # 推当前分支
* git push -u origin feature/x # 建立跟踪关系
* git push --tags # 推送所有标签

**6) 标签与发布**

**git tag / git describe**

* **作用**：打版本点、发布标记。
* **用法**：
* git tag v1.0.0
* git tag -a v1.0.1 -m "热修复"
* git push origin v1.0.1
* git describe --tags # 显示距离最近标签的描述

**7) 临时存货：git stash**

* **作用**：把未提交改动临时“塞进抽屉”，工作区变干净。
* **常用**：
* git stash push -m "进行中的报表导出" # 保存
* git stash list
* git stash pop # 取出并删除
* git stash apply stash@{2} # 取出但保留
* git stash -p # 只存部分

**8) 撤销与回滚（从安全到危险）**

**A. 最安全：git revert**

* **作用**：制造一笔“反向提交”来撤销某个历史提交，不改历史。
* **用法**：
* git revert <commit-id>

**B. 中等风险：git restore（撤工作区/暂存）**

见上文。

**C. 危险但常用：git reset**

* **作用**：重设当前分支的指向。
* **用法**：
* git reset --soft HEAD^ # 回退到上一提交，改动全回到暂存
* git reset --mixed HEAD^ # 默认：回到工作区（未暂存）
* git reset --hard HEAD^ # ⚠️ 全丢弃（慎用）
* **常见场景**：误提交到本地还没 push，--soft/--mixed 很香。

**D. 绝望时的救命绳：git reflog**

* **作用**：看“HEAD 曾经指向过哪里”，可找回误删分支/重置的提交。
* **用法**：
* git reflog
* git switch -c rescue <安全点的commit-id>

**9) 精准移植：git cherry-pick**

* **作用**：把一两个提交“挑拣”到当前分支。
* **用法**：
* git cherry-pick <commit-id>
* git cherry-pick A^..B # 区间（包含 A 之后到 B）

**10) 清理与维护**

**git clean**

* **作用**：删除未跟踪文件（不在版本库里的）。
* **用法**：
* git clean -n # 仅预览
* git clean -fd # 目录也删（危险）

**git rm / git mv**

* **作用**：删除/重命名已跟踪文件（带入版本历史）。
* **用法**：
* git rm path/file
* git mv old new

**git gc**

* **作用**：垃圾回收、瘦身。
* **用法**：git gc

**11) 历史改造（本地私有分支用）**

**交互式变基：git rebase -i**

* **作用**：合并、小改名、重排提交。
* **用法**：
* git rebase -i HEAD~5
* # 把 pick 改成 squash/fixup/reword 等

**修补信息：git commit --amend**

* **作用**：修改最近一次提交（信息或内容）。
* **用法**：
* git add <更多文件>
* git commit --amend --no-edit

**生成补丁 / 应用补丁：git format-patch / git apply / git am**

* **用法**：
* git format-patch -1 <commit> -o patches/
* git apply patches/0001-\*.patch
* git am patches/0001-\*.patch

**12) 子模块与稀疏检出**

**git submodule**

* **作用**：在项目中引入另一个仓库。
* **常用**：
* git submodule add <repo-url> vendor/libA
* git submodule update --init --recursive
* git submodule foreach git pull --rebase

**git sparse-checkout**

* **作用**：只检出仓库的部分目录，巨仓库省磁盘/提速。
* **快速上手**：
* git sparse-checkout init --cone
* git sparse-checkout set src/backend docs

**13) 调试定位：git bisect**

* **作用**：二分查找引入 bug 的提交。
* **流程**：
* git bisect start
* git bisect bad # 当前是坏的
* git bisect good <good-commit> # 早期好的点
* # Git 会切到一个中间点 -> 你测试好坏 -> 标记：
* git bisect good|bad
* git bisect reset # 结束恢复

**14) 常用别名（懒人必配 🧰）**

git config --global alias.s "status -sb"

git config --global alias.lg "log --oneline --graph --decorate --all"

git config --global alias.co "checkout"

git config --global alias.sw "switch"

git config --global alias.cm "commit -m"

git config --global alias.st "stash"

git config --global alias.fp "fetch -p"

git config --global alias.rbi "rebase -i"

git config --global alias.last "show -1 --stat"

git config --global alias.unstage "restore --staged --"

git config --global alias.undo "reset --soft HEAD~1"

**15) .gitignore & .gitattributes（最常用片段）**

**.gitignore 常见：**

# Node / JS

node\_modules/

dist/

\*.log

# Python

\_\_pycache\_\_/

\*.pyc

.venv/

# IDE

.vscode/

.idea/

# OS

.DS\_Store

Thumbs.db

**.gitattributes（统一换行符/合并策略示例）：**

\* text=auto

\*.sh text eol=lf

\*.bat text eol=crlf

**场景化工作流（拿去即用）**

**A) 新建项目并首推**

git init

git branch -M main

git add -A

git commit -m "chore: init project"

git remote add origin <repo-url>

git push -u origin main

**B) 日常开发（保持历史干净）**

git fetch origin

git switch -c feature/export-report origin/main

# 开发...

git add -A

git commit -m "feat: 导出报表为 CSV"

git pull --rebase # 同步远程并把我的提交平铺到最新

git push -u origin HEAD

**C) 代码评审前整理提交**

git rebase -i origin/main # squash/fixup 小提交

git push --force-with-lease # 只在自己的功能分支使用

⚠️ 公共分支不要改历史！--force-with-lease 比 --force 安全，能防止覆盖别人。

**D) 修复线上 bug（hotfix）**

git fetch origin

git switch -c hotfix/timeout origin/main

# 修复...

git add -A

git commit -m "fix: 请求超时处理"

git push -u origin HEAD

# 走 PR -> 合并 -> 打标签发版

git tag -a v1.0.2 -m "hotfix: timeout"

git push origin v1.0.2

**E) 临时切出审查别的分支，但不想提交未完成工作**

git stash push -m "WIP: 导出进度"

git switch someone/branch

# 看完回来

git switch feature/export-report

git stash pop

**F) 提交写错了 / 不该包含某文件**

git restore --staged path/secret.txt # 从暂存区撤回

git restore path/secret.txt # 撤销工作区改动

# 或者上一提交撤回：

git reset --soft HEAD~1

# 重新添加想要的文件再提交

**G) 合并冲突处理套路（无脑模板）**

git pull --rebase

# 如果冲突：

# 1) 打开文件，按 <<<<<<< / ======= / >>>>>>> 标记手动修

git add <冲突文件们>

git rebase --continue # 如果用 merge：git commit

# 实在乱了：

git rebase --abort # 或 merge --abort

**H) 找回“误删分支/误 reset”的历史**

git reflog # 找到还在的 commit id

git switch -c rescue <id> # 拉一条救援分支

**I) 只拉巨仓库的一角（稀疏检出）**

git clone <repo>

cd repo

git sparse-checkout init --cone

git sparse-checkout set services/api docs

**J) 从 A 分支“挑”一个修复到 B 分支**

git switch release/1.0

git cherry-pick <fix-commit-id>

**常见报错速译 & 解法 🩹**

* **“fatal: refusing to merge unrelated histories”**  
  两个仓库历史不相干。
* git pull origin main --allow-unrelated-histories
* **“Updates were rejected because the tip of your current branch is behind”**（远程领先）
* git pull --rebase
* git push
* **“non-fast-forward” 且你确定只你改过（自己的功能分支）**
* git push --force-with-lease
* **“detached HEAD”**（游离 HEAD）  
  你在某个提交上而不是分支上：
* git switch -c temp-fix

**团队建议（不学理论也能不踩坑）🧭**

* main/develop 等公共分支：**禁止 --force**、禁止改历史。
* 个人分支：随便整理历史（rebase -i / --amend），但合并前确保同步 origin/main。
* 统一开启：
* git config --global pull.rebase true
* 推前自检：git status → git diff --staged → git log -1。
* 冲突不慌：先 git status 看冲突文件，再逐一解决并 git add，最后 --continue。
* 危急时刻：先 git reflog，再动刀。

**进阶但超有用的几味小料 🍜**

* **git worktree**：在同一仓库同时修两个需求，丝滑。
* **git bisect**：自动帮你二分找到引入 bug 的那一笔。
* **git describe**：快速知道“我现在距离哪个版本最近”。
* **git grep**：比 IDE 更快的全库搜索。