按环境清单、镜像和参数记录演练历史任务复现

完整操作教程 · 参考固定 · 往返恢复验收

论文返修需要重跑旧分析时，单独保存一个包版本号通常不够。本文说明如何把输入、脚本、参数、环境和运行证据组成一份可审计的复现清单。

运行边界：manifest仅用于记录；renv不会锁定整个操作系统，容器也不能固定所有宿主机依赖。

先区分“恢复环境”和“复现结论”

半年后重新运行一个分析，至少涉及四类对象：当时的输入数据、代码与参数、软件环境、运行资源和系统依赖。恢复R包版本只能解决其中一部分。renv可以记录和恢复项目R包环境，但项目仍可能依赖R本身、操作系统库、编译器、系统字体、外部命令行软件、参考数据库和环境变量。容器可以固定更多系统依赖，但容器标签可能被重新指向；使用镜像摘要比只记录可变标签更容易锁定同一镜像。即便环境相同，随机数、并行算法、硬件数学库和输入文件变化仍可能导致结果差异。

因此应先问清论文返修的目标：需要重做相同图表、补充一个分析步骤、还是确认结论对参数变化是否稳健？如果目标只是恢复某张图，保存代码、输入版本、参数和图形依赖可能足够；若需要严格重跑整条流程，还需要所有中间文件、参考库、脚本入口、资源限制和输出检查。历史输入如果已按计划删除，应如实记录无法复现的部分，不要用名称相似的新数据冒充原文件。

建立一次运行的材料清单

每次正式分析在开始前生成manifest：项目标识、运行ID、输入文件相对路径与SHA256、脚本仓库提交、工作目录、完整参数、软件与外部工具版本、R/Python版本、容器镜像摘要、参考数据库版本、随机种子、线程数、CPU/内存/GPU资源、开始结束时间、退出码和输出文件摘要。记录来源而不复制全部敏感数据；如果数据受限，保留哈希和受控存储位置，并确保将来仍能获授权读取。

运行结束后，保存sessionInfo或conda list只是环境快照的一部分。还要保存执行命令、标准错误摘要、失败重试情况以及关键输出的结构验收。例如单细胞流程可以核对细胞数、基因数、样本列、过滤规则、降维维数和图文件哈希；BLAST流程则保存数据库manifest和软件参数。不要把日志中可能存在的样本名、临床变量或令牌无差别公开。

层次	应记录的证据	常见遗漏
输入	路径、授权位置、SHA256、版本	只保存文件名
代码	Git commit、脚本与参数	本地未提交修改
环境	lockfile、镜像摘要、系统依赖	只写“R 4.x”或latest标签
运行	资源、随机种子、命令、退出码	只保存成功输出
结果	摘要、结构检查、图表版本	没有记录图对应的输入

renv与容器各自能解决什么

在R项目中，可将renv.lock纳入版本控制，并在隔离目录恢复。先复制项目到新路径，检查锁文件中记录的R版本和包来源，再使用renv::restore恢复包。安装失败时区分：仓库中的二进制包不可用、旧源码需要系统依赖、R版本不兼容、私有包源不可访问。不要在唯一的生产库上直接升级包来解决旧项目恢复；另建项目库并记录尝试结果。

renv锁定的是R包依赖，不会自动冻结操作系统或所有外部程序。容器可封装R和系统库，但要记录不可变镜像摘要、Dockerfile或构建配方、构建时间和基础镜像来源。镜像内部若从CRAN安装“当前最新版”，重建镜像仍可能改变依赖；要将包锁文件纳入镜像构建。Rocker项目也提示需要精确镜像时使用digest，而不是仅依赖tag。若有显卡、GPU驱动、

CUDA或外部挂载数据，容器并不能单独固定宿主机驱动和数据内容。

示例：制作运行manifest并校验输入

下面的脚本仅展示如何把重要字段和输入摘要写成JSON。将路径限制在项目授权目录，避免递归扫描共享根目录。SHA256计算大型文件会读取完整文件，可能产生明显I/O；正式任务可在文件落盘时同步计算，或使用已有可信校验结果。manifest本身不保存密钥，也不等同于软件环境锁定。

```
import hashlib, json, platform, subprocess
from pathlib import Path
inputs = [Path("data/counts.tsv"), Path("config/analysis.yaml")]
def sha256(path):
    h = hashlib.sha256()
    with path.open("rb") as f:
        for block in iter(lambda: f.read(1024 * 1024), b""):
            h.update(block)
    return h.hexdigest()
manifest = {
    "run_id": "revision-demo-001",
    "code_commit": subprocess.check_output(
        ["git", "rev-parse", "HEAD"], text=True).strip(),
    "python": platform.python_version(),
    "platform": platform.platform(),
    "random_seed": 101,
    "threads": 4,
    "inputs": [{"path": str(p), "sha256": sha256(p)} for p in inputs],
    "container_digest": "RECORD_IMAGE_SHA256_HERE",
    "parameters_file": "config/analysis.yaml",
}
Path("run-manifest.json").write_text(
    json.dumps(manifest, indent=2), encoding="utf-8")
```

脚本假设Git仓库干净且输入文件存在；正式版本还应记录`git status --porcelain`，并在有未提交更改时明确保存补丁或阻止生成“可复现”标签。镜像摘要必须从实际执行环境读取，不能用占位字符串。输入路径应使用相对路径或脱敏标识，项目目录结构变化时要保留映射。并行线程数需要与调度器实际分配的资源一致。

在隔离副本中恢复并做结果比较

复现不要直接覆盖论文最终结果目录。创建新的run_id和输出路径，复制或只读挂载输入，恢复锁文件或拉取镜像摘要，再执行同一命令。先做环境检查和小样本dry run，确认依赖、权限和存储后再全量运行。结束后比较输入SHA、参数、版本、输出结构和关键数值。完全相同的哈希可证明文件字节相同；图像哈希不同可能只来自元数据或绘图字体变化，不宜立即判定科学结论变化。对数值结果可预先设定合理容差和关键指标，不要看到差异后再挑对自己有利的比较口径。

结果不一致时按顺序排查：输入摘要是否一致，脚本是否有未提交修改，参考数据库是否换版，系统库或线程是否变化，随机数生成器是否被全部设置，外部服务或API是否返回新数据。将差异原因和处理过程记录在返修说明中。可复现环境提高的是过程可追踪性，并不自动证明原分析设计合理，也不保证相同结果必然成立。

归档和交接避免单人依赖

最终归档应有README、运行manifest、lockfile或镜像digest、代码commit、输入授权位置、命令、输出清单、QC摘要和责任人。另设一份精简的复现说明，写明哪些步骤可离线完成、哪些参考库需申请访问、哪些文件不可公开。敏感数据不必打进可下载的环境包；复现包可以包含合成小样本和结构模板。

团队至少让第二名成员按说明在隔离目录完成一次小规模恢复，记录遇到的人工步骤。若必须依赖某位成员家目录中的缓存或私有秘钥，说明还不完整。服务器环境可以提供稳定的运行目录、版本化容器和足够的CPU/存储，但具体复现要基于项目实际材料判断；若旧输入、参考库或外部许可缺失，先补齐可用性再评估资源。

用演练检验材料齐不齐，不靠文档自我证明

一份清单只有经过第二人尝试恢复，才知道是否缺少隐含步骤。先从归档副本建立新的工作目录，断开原来用户的个人缓存和历史工作区，按README执行。记录每一步耗时、访问请求和失败原因。若恢复只能在原作者家目录成功，说明依赖路径未被记录；若必须手工下载浮动版本的数据库或软件，说明环境仍会随时间变化。对不允许联网的受限数据，可提前准备已审核的依赖包和checksum，按单位流程离线导入。

演练可分三层：环境启动层检查R/Python版本、系统库和外部程序；小样本运行层检查输入schema、命令、随机种子和输出结构；全量恢复层才评估资源与运行时间。小样本演练要能发现样本列名、依赖数据库和输出目录约定等问题，但不能拿小样本结果替代全量分析结论。演练失败时分类记录为“缺材料”“版本不可用”“权限不足”“资源不足”或“结果不一致”，分别采取补材料、找归档、申请授权或重新审查方法等动作。

精确环境恢复与科学复现也要区分。字节级完全一致可能受压缩元数据、文件系统顺序、线程调度、BLAS库和绘图字体影响；浮点计算在不同硬件或并行设置下可能有微小差异。应在运行前确定什么结果必须严格一致、什么结果允许数值容差、哪些图表需要人工核查。不要在看到差异后才定义“足够接近”。对关键结论，比较预先指定的统计量、候选排序、置信区间或分群稳定性，并说明判断依据。

软件包之外的依赖也要归档

分析常依赖Ensembl/GENCODE注释、基因组FASTA、数据库下载、外部脚本和硬件加速库。参考库需要记录来源、版本和摘要；若许可证不允许打包分发，记录申请路径与取得方式。某些R包从GitHub安装时还需记录commit或release，而不是只记包名。系统依赖如GDAL、HDF5、字体、Java、编译器和CUDA可能影响安装与运行；容器能封装一部分，但宿主驱动、挂载卷和调度器资源仍在容器之外。

容器镜像要保存实际digest和构建文件。若镜像只使用`:latest`或可移动tag，数月后重新拉取可能得到不同内容。镜像拉取成功也不证明输入数据、宿主卷或数据库版本一致。保存构建配方可以解释镜像如何来，但如果基础软件仓库不可用，仍不一定能重建。重要项目可在授权的镜像仓库保留digest对应的镜像副本，并记录镜像存储位置和访问权限。

对于renv，确认lockfile包含所有项目直接依赖，且本地未使用未记录的开发包。恢复前先检查项目R版本、仓库配置、包源和系统依赖。若旧版本无法在当前操作系统编译，不要擅自升级后继续声称复现了原环境；可以用隔离容器、归档二进制包或记录兼容性差异。任何手工修复都写入新的补充记录，保留原始lockfile不覆盖。

复现过程不能泄露密钥或原始数据

环境包和manifest应排除API令牌、数据库密码、SSH私钥、未经许可的数据和个人路径。代码库提交前扫描配置文件和历史提交；仅从当前文件删掉密码不代表Git历史已经清除。若曾将密钥提交到版本库，应按机构流程撤销并轮换密钥。对外共享的复现包可以使用小型合成输入和固定目录结构，让对方验证流程能启动，再通过受控渠道访问真实输入。

压缩包里的文件要清点、哈希并说明许可。大型参考数据库可以单独列出校验值，不必重复打包；接收方应能验证文件版本和完整性。受限数据的哈希本身一般不会直接暴露内容，但样本命名、项目号和路径仍可能具有敏感性。公开附件前由项目负责人审核清单，而非只由技术人员确认压缩包能打开。

形成返修可用的证据链

最终复现记录应能回答：旧输入是否可取得且摘要一致；用什么代码和参数；环境如何恢复；在哪些硬件与资源约束下运行；程序退出状态如何；输出与旧结果比较了什么；差异如何处理。论文返修可以引用这套证据说明方法与版本，但不能把“代码成功运行”写成结论获得独立验证。若旧环境不可恢复，要明确指出缺少的文件、许可证或参考库，并说明补救方案及其对结果解释的影响。

半年以后能不能重跑，最终取决于团队当时是否保存了足够材料、数据授权是否仍有效、依赖是否可获取，以及结果验收标准是否明确。服务器或容器可以提供稳定的计算底座，但不能替代输入归档、版本管理和研究记录。先做一次小规模恢复演练，通常比等返修期限临近才发现个人缓存或旧数据库已经消失更稳妥。

计算资源变化时，记录性能和结果两类验收

旧项目在新服务器上运行，即便软件环境相同，CPU型号、内存、存储吞吐、线程库和GPU驱动仍可能不同。先使用小规模代表性数据估算峰值资源和运行时长，再根据完整任务安排CPU、内存、存储与并行度。记录实际分配资源而不只记录请求参数，因为调度系统可能改变可用配置。检查临时目录、scratch和输出盘是否分属不同文件系统；满盘可能造成文件截断或中间文件丢失。任务前确认备份与恢复责任，重要的历史结果不要在第一次重跑时覆盖。

性能验收与科学结果验收分开写。性能部分比较耗时、CPU利用率、峰值RSS、IO和资源退出状态；科学结果部分比较预先指定的统计量、排序、分组或图形结构。新机器上运行更快不说明结论正确，运行更慢也不必然说明环境错了。若新旧平台结果略有差异，先检查线程数、数学库、随机数生成方式、数据读写精度和并行归约顺序。允许的数值误差应由方法和数据特点确定，而不是统一套一个很宽的阈值。

返修期间若脚本或参数需要调整，保留原始运行manifest并创建新的commit与run_id。记录变更原因和影响范围，明确哪些图表由旧版本产生、哪些由新版本产生。如果只是修复绘图标签或排版，也要让图与计算结果版本对应。科研团队可采用“探索运行”“确认运行”“归档运行”三个标签，避免把临时试错结果误交给合作者或写进论文。

环境迁移前应先核对许可证、数据库访问权限和受控数据位置。镜像复制到新服务器并不意味着数据也可以随之复制；使用外部存储时要确认挂载权限和备份策略。若历史数据因项目政策不能保留，需记录归档责任人和重新获取方式，并在论文复现说明中披露限制。合规边界优先于技术上的完整打包。

配套练习包

本PDF对应的ZIP包含本文代码、主题模板或合成数据、README和SHA256校验文件。运行前核对包内清单，并按文章说明在独立目录验证。

参考资料

renv restore 官方文档 · 来源资料

Rocker 环境复现说明 · 来源资料

Kubernetes 镜像摘要文档 · 来源资料

操作建议：先在合成小样本上验证流程，再依据项目参考、标签合同、存储策略与获配资源验收真实数据。