AnnotSV批量注释教程：从VCF检查到结果验收

以GRCh38结构变异VCF为例，逐步整理输入清单、固定AnnotSV注释资源、运行full模式，并检查过滤范围、输出行和失败任务。

同一批结构变异即使都保存为VCF，也可能因参考组装、调用器字段和事件表示不同而不能直接比较。本文以GRCh38上的DEL、DUP VCF为例，演示如何批量运行AnnotSV，并保留足够信息让组内成员复查。BND、复杂重排和非人类数据应先确认对应版本的输入要求，不要直接套用此例。

阶段	检查内容	留下的记录
输入	参考组装、样本、SV字段与过滤	VCF头、散列、调用器版本
运行	固定AnnotSV与注释库	命令参数、日志、耗时和峰值RSS
验收	表头、full模式及事件对应关系	成功、过滤、未解释记录清单

运行前先固定输入和注释资源

准备一个无表头的清单sv-inputs.tsv，每行两列，以TAB分隔：唯一任务ID、VCF绝对路径。例如：

```
case01■/data/sv/case01.vcf.gz
case02■/data/sv/case02.vcf.gz
```

先确认VCF样本名、FILTER、SVTYPE、END或SVLEN等字段，并记录调用器及版本。检查参考组装和染色体命名与AnnotSV注释资源一致；不要为适配脚本而随意改POS。VCF和BED坐标规则不同，若必须转换，单独用已知事件核对边界。

AnnotSV程序和注释目录都要记版本。以下命令假设注释资源已按官方说明准备在固定目录；更新时使用新目录，不要在批次运行中覆盖原目录。AnnotSV的-SVminSize默认值为50 bp，示例显式写出该阈值，意味着较小事件可能不会进入注释，需按研究设计调整。

逐个运行并保存日志

在工作目录新建空的annotsv-batch文件夹。下面脚本为每个ID建立独立目录，保存输入校验值、VCF头、样本名、耗时及标准输出；先串行跑通，再根据实际峰值资源增加并发。

```
set -euo pipefail
annotations=/ref/AnnotSV-annotations-fixed
test -d "$annotations"
mkdir annotsv-batch
AnnotSV -version > annotsv-batch/version.txt
declare -A seen=(); n=0
while IFS=$'\t' read -r id input extra || [[ -n "${id:-}${input:-}${extra:-}" ]]; do
  [[ "$id" =~ ^[A-Za-z0-9][A-Za-z0-9_.-]*$ ]]
  [[ "$input" = /* && -z "${extra:-}" ]]
  [[ -z "${seen[$id]+yes}" ]]; seen[$id]=1
  test -s "$input"
  d="$PWD/annotsv-batch/$id"; mkdir "$d"
  bcftools view -h "$input" > "$d/input.header.txt"
  bcftools query -l "$input" > "$d/input.samples.txt"
  sha256sum "$input" > "$d/input.sha256"
  /usr/bin/time -v -o "$d/time.txt" AnnotSV -SVinputFile "$input" \
    -genomeBuild GRCh38 -annotationsDir "$annotations" \
    -annotationMode full -SVminSize 50 -overwrite 0 \
    -outputFile "$d/annotated.tsv" > "$d/run.log" 2>&1
  test -s "$d/annotated.tsv"
  python3 check-annotsv.py "$d/annotated.tsv" > "$d/table-check.txt"
  printf 'COMPUTE_OK\n' > "$d/status.txt"
  n=$((n+1))
done < sv-inputs.tsv
test "$n" -gt 0
```

检查TSV和事件覆盖

把下面检查器保存为check-annotsv.py。它确认输出包含AnnotSV ID与模式列、每行字段完整且模式为full；合法的只有表头、没有数据行的文件也会通过，因此还要结合输入过滤和运行日志解释零结果。检查器不证明每条输入都成功注释。

```
import csv, sys
with open(sys.argv[1], newline="") as f:
    r = csv.DictReader(f, delimiter="\t")
    need = {"AnnotSV_ID", "Annotation_mode"}
    if not r.fieldnames or not need.issubset(r.fieldnames): raise ValueError("required
columns missing")
    rows = list(r)
for row in rows:
    if None in row or any(v is None for v in row.values()) or not row["AnnotSV_ID"] or
row["Annotation_mode"] != "full":
        raise ValueError("malformed or unexpected row")
print("full annotation rows:", len(rows))
```

随后按原始ID列、AnnotSV_ID及样本字段核对输入输出对应关系；不要假设软件生成的AnnotSV_ID一定等于原始VCF ID。分别记录输入记录数、因阈值或过滤未进入的记录、成功输出及未解释记录。不要只比较总数：两个输入可能有相同条数却缺失了不同事件。将未匹配ID单独导出，检查是否为过滤、ID重写或未完成任务，再结合FILTER、SVTYPE、SVminSize和日志逐项归因；原因未明时保留为待核查，不要静默丢弃。full用于事件层面的整行注释；split可能按基因或区域展开多行，both则包含两种模式。因此不能用TSV行数直接代表SV数，需按ID和Annotation_mode分别汇总。

结果、异常与资源验收

抽查若干DEL/DUP：参考版本、染色体、起止位置、样本、重叠基因和注释来源是否符合预期。AnnotSV还可使用VCF中的断点置信区间扩展位置（-includeCI默认开启）；记录该设置，避免不同批次的重叠结果因配置不同而被误判为生物学差异。注释和排序提供候选线索，不能替代独立证据审查或临床分类。

失败时先看退出码和run.log，再检查VCF解析、参考版本、注释资源路径及权限、磁盘空间和内存。保留失败目录，修复原因后用新的尝试目录重跑，避免旧TSV被误当成新结果。记录程序与注释库版本、参数、输入散列、耗时、峰值RSS及状态；只有代表性任务确实受计算资源限制时，再据实安排CPU、内存和工作盘。

参考资料

AnnotSV v3.5.10 命令行参数（官方） 官方文档
AnnotSV 官方项目与安装说明 官方文档