ОБЪЕДИНЁННЫЙ ИНСТИТУТ ЯДЕРНЫХ ИССЛЕДОВАНИЙ (ОИЯИ) • ЛИТ & ЛЯП

# Вычислительный кластер ОИЯИ: Нейтринная платформа (NCP)

Пошаговое руководство для новых пользователей: получение доступа, архитектура кластера, правила работы с данными и запуск вычислительных задач через систему очередей HTCondor

## ПАСПОРТ ПЛАТФОРМЫ

Точка входа (Submit-узел):
`ssh login@10.220.18.43`

Пакетный планировщик:
**HTCondor (Batch System)**

Хранилища и среда:
CephFS (/taiga, /juno, /nova)
CVMFS + Apptainer, EOS

Поддерживаемые проекты:
**TAIGA, Baikal-GVD, JUNO, NOvA, DUNE, Borexino и др.**

**ГЛАВНОЕ ПРАВИЛО НЕЙТРИННОЙ ПЛАТФОРМЫ: ЗАПРЕТ ПРЯМОГО ЗАПУСКА РАСЧЁТОВ!**

На входном узле (10.220.18.43), куда вы попадаете по SSH, категорически нельзя запускать расчётные скрипты и моделирование напрямую в терминале! Входной узел предназначен только для сборки кода и управления очередью.

**Все вычисления должны запускаться строго через менеджер очередей HTCondor (команда condor_submit).**

На основе официальной документации системных администраторов NCP (git.jinr.ru/neutrino_group/bootcamp) и руководства группы TAIGA (git.jinr.ru/taiga/common)

# Что такое Нейтринная платформа ОИЯИ и почему нельзя считать напрямую

Как устроена коллективная работа на вычислительном кластере и зачем нужна пакетная очередь HTCondor

## Назначение Нейтринной платформы (NCP)

- **Единый вычислительный пул ОИЯИ:** создан совместно ЛИТ и ЛЯП ОИЯИ в составе МИВК для обработки данных и Монте-Карло моделирования.
- **Участники платформы:** эксперименты TAIGA, Baikal-GVD, JUNO, NOvA, DUNE, Borexino, Daya Bay, DarkSide, GNA.
- **Пакетная обработка (Batch System):** пользователь не ищет свободный сервер вручную — он описывает задачу (Job), а система HTCondor сама находит свободные ядра и память в пуле.
- **Масштабируемость:** вместо одного долгого расчёта вы можете параллельно запустить сотни и тысячи независимых задач на десятках рабочих серверов.

## Почему НЕЛЬЗЯ запускать скрипты просто так?

- **Submit-узел (10.220.18.43) — один на всех!** По SSH вы попадаете на общий входной сервер, где одновременно работают десятки коллег.
- **Крах общей работы при прямом счёте:** запуск расчётов (./my_sim, Geant4, CORSIKA, Python) прямо в SSH-сессии исчерпывает память и CPU входного узла и блокирует терминалы всем пользователям.
- **Что РАЗРЕШЕНО на submit-узле:** редактирование кода, git, сборка программы (в контейнере), короткие тесты и отправка задач (condor_submit).
- **Что ОБЯЗАТЕЛЬНО отправлять в HTCondor:** любое моделирование, обработку данных и ресурсоёмкие расчёты.

## Схема взаимодействия с платформой: НЕПРАВИЛЬНО vs ПРАВИЛЬНО

**НЕПРАВИЛЬНО (Нарушение правил)**

**Ноутбук —(SSH)→ Submit-узел (10.220.18.43) → Прямой запуск ./run_sim.sh в консоли**

Последствия: перегрузка CPU/RAM входного узла, зависание очереди, принудительная остановка процессов администратором.

**ПРАВИЛЬНО (Стандарт NCP)**

**Ноутбук —(SSH)→ Submit-узел (run.sh + job.sub) —(condor_submit)→ Рабочие узлы (Worker Nodes)**

Преимущества: расчёт идёт на выделенных ядрах с быстрым локальным диском (scratch), а готовые данные сохраняются в CephFS.

# Где живут данные и где выполняются вычисления

Три ключевых элемента инфраструктуры NCP: входной узел, вычислительные узлы и распределённые файловые системы

## 1. Submit-узел (Входной узел)

`IP: 10.220.18.43  (10-220-18-43)`

- **Точка входа по SSH** для всех пользователей платформы.
- **Подготовка кода:** клонирование репозиториев из git.jinr.ru, компиляция программ внутри контейнера Apptainer.
- **Управление задачами:** отправка заданий (condor_submit), проверка очереди (condor_q) и истории (condor_history).
- **Важно:** домашний каталог (~/) submit-узла НЕ виден автоматически на рабочих узлах!

## 2. Рабочие узлы (Worker Nodes)

**Динамические слоты HTCondor**

- **Ресурсы ОЗУ:** минимум 4 ГБ RAM на ядро, на крупных узлах — до 14 ГБ RAM на ядро.
- **Локальный диск (Scratch):** около 20 ГБ на ядро. Путь задаётся переменной $_CONDOR_SCRATCH_DIR.
- **Изоляция задачи:** каждая задача получает свой временный каталог scratch, который автоматически очищается после завершения.
- **Доступ к сетям:** видят общие диски CephFS, CVMFS и хранилище EOS.

## 3. Хранилища и ПО (Storage)

**CephFS • CVMFS • EOS (XRootD)**

- **CephFS (общие директории):** доступны по одинаковым путям на submit и worker узлах: /taiga, /baikalgvd-nova, /juno, /nova, /dune, /borexino, /mnt/cephfs.
- **CVMFS (/cvmfs/...):** глобальный каталог готового научного софта и образов контейнеров Singularity/Apptainer.
- **EOS (root://eos.jinr.ru):** массовое хранилище данных; работа через протокол XRootD (xrdcp, xrdfs).

## Стандартный цикл движения данных при расчёте на Нейтринной платформе

**1. Входные данные в CephFS**
Файлы и скрипты лежат в /taiga или /juno

→

**2. Копирование в Scratch**
cp в $_CONDOR_SCRATCH_DIR на рабочем узле

→

**3. Быстрый локальный счёт**
Чтение/запись временных файлов на локальном диске

→

**4. Публикация результата**
Атомарное сохранение итога обратно в CephFS

# Как получить доступ на Нейтринную платформу (План действий)

Пошаговая инструкция оформления учётных записей ОИЯИ, доступа в CICC и включения в группу проекта

### ШАГ 1: Почта @jinr.ru и SSO-аккаунт

- **Где оформить:** в ЛИТ ОИЯИ в сетевом отделе (комн. 200) или через сайт NOC:

  `https://noc.jinr.ru/ru/registry/index.php`

- **Зачем нужен:** единый вход (SSO) на веб-ресурсы ОИЯИ (pin.jinr.ru, disk.jinr.ru, git.jinr.ru).
- **Важно помнить:** одного только SSO-аккаунта НЕДОСТАТОЧНО для входа по SSH на кластер!

### ШАГ 2: Вычислительный аккаунт (CICC)

- **Заполнить электронную анкету CICC:**

  `lxs-s03.jinr.ru/cicc/index.php/en/cicc-resource-request/`

- **Что указать в анкете:** в графе Resources выбрать ВСЕ пункты, вписать свой email в домене @jinr.ru.
- **Отправить анкету:** написать Наталье Громовой (grom@jinr.ru) для создания компьютерного аккаунта.

### ШАГ 3: Проверка email и группа проекта

- **Проверить профиль SSO:** зайти на login.jinr.ru → Personal profile → Change E-mail и убедиться, что вписан актуальный Primary e-mail.
- **Добавление в группу:** попросить Н. Громову (grom@jinr.ru) добавить ваш аккаунт в Linux-группу эксперимента (например, taiga или juno).
- **Права на папки:** без группы проекта не будет доступа к /taiga или /juno.

### ШАГ 4: Доступ к сети, GitLab и CVMFS

- **Доступ к сети и сабмит-узлу:** при работе вне сети ОИЯИ настройте VPN / SSH-шлюз по инструкции проекта.
- **Публикация софта в CVMFS:** передайте публичную часть своего SSH-ключа (~/.ssh/id_ed25519.pub) действующим участникам проекта.
- **GitLab ОИЯИ:** зарегистрируйтесь на git.jinr.ru с почтой @jinr.ru для доступа к вики и коду.

### Справочные реквизиты для заполнения анкеты (из инструкции проекта TAIGA)

- **Номер темы ОИЯИ (для анкеты):** 02-2-1099-3-2015/2026
- **Руководитель темы:** Бородин А.Н. (borodin@jinr.ru)
- **Оформление аккаунта в ЛИТ:** Громова Наталья (grom@jinr.ru)
- **Координатор по доступу в TAIGA:** Александр Блинов — Telegram @Alspy
- **Инструкция CICC ОИЯИ:** lxs-s03.jinr.ru/cicc/index.php/registration_at_cicc-ru/
- **Проверка номера темы:** plan.jinr.ru/2026/37-blog-2026/1700-02-0-1065-2007-2023

# Первый вход на Submit-узел и сборка программ в контейнерах

Как проверить свою учётную запись, собрать программу в совместимом окружении Apptainer и не перепутать режимы

## 1. Подключение по SSH и проверка групп

- **Вход на submit-узел:** используйте логин и пароль от вычислительного аккаунта (или SSH-ключ).
- **Обязательная проверка:** убедитесь, что доступен HTCondor и ваш аккаунт входит в группу проекта (taiga).

**Терминал: подключение и самопроверка**

```
$ ssh your_login@10.220.18.43   # или 10-220-18-43
$ condor_version                # проверка HTCondor
$ id -nG                        # список ваших групп
your_login taiga               # <-- группа должна быть в списке!
```

## Важно: Linux-группа vs accounting_group

- **Членство в Linux-группе (id -nG):** даёт реальные права файловой системы на чтение и запись в общие каталоги проекта на CephFS (/taiga, /juno, /nova).
- **Параметр accounting_group = taiga:** указывается в файле задачи job.sub для учёта потреблённых процессорных часов за вашим проектом в статистике HTCondor.
- **Одно не заменяет другое!** Если после добавления в группу её нет в выводе id -nG, переподключитесь по SSH заново.

## 2. Подготовка софта и контейнеры Apptainer / CVMFS

- **Совместимость окружений:** программа должна собираться и выполняться с одними и теми же версиями библиотек (ROOT, Geant4 и др.) и работать без интерактивного ввода.
- **Готовые контейнеры в CVMFS:** образы лежат в /cvmfs/containers.jinr.ru/... и видны на всех узлах. Свой образ публикуется через git.jinr.ru/cvmfs-container-sync/cvmfs-singularity-sync.
- **Как собрать программу на submit-узле:** откройте shell внутри контейнера, скомпилируйте проект и обязательно выйдите (exit) обратно на узел!

**Сборка софта внутри контейнера Geant4 на submit-узле**

```
# 1. Запускаем интерактивный shell внутри контейнера CVMFS:
$ apptainer exec -B /taiga -B /cvmfs \
  '/cvmfs/containers.jinr.ru/taiga/containers/taiga_g4:el9' /bin/bash
# 2. Внутри контейнера: подключаем окружение и собираем код:
Apptainer> mkdir -p build && cd build && cmake .. && make -j4
# 3. ОБЯЗАТЕЛЬНО выходим из контейнера перед отправкой задач!
Apptainer> exit
```

**ВАЖНОЕ ПРАВИЛО:**

Команды condor_submit и condor_q вызывайте ТОЛЬКО из обычного shell submit-узла (после exit из контейнера). Настройку окружения (source ...) обязательно включите внутрь скрипта запуска задачи!

# Из чего состоит задача в HTCondor: два обязательных файла

Чтобы отправить расчёт на кластер, вы всегда готовите пару файлов: исполняемый скрипт и паспорт задачи

## Файл №1: Скрипт расчёта (run.sh / example_script.sh)

- **Что это такое:** обычный bash-скрипт (#!/bin/bash), который HTCondor запустит на выбранном рабочем узле (внутри контейнера или напрямую).
- **Права на запуск:** обязательно сделайте скрипт исполняемым командой:

chmod +x run.sh

- **Что должно быть внутри скрипта:**
  1. `Строгий режим ошибок: set -euo pipefail`
  2. Приём аргументов: ClusterId, ProcId, пути к данным
  3. `Переход в локальный scratch: cd "$_CONDOR_SCRATCH_DIR"`
  4. Подключение окружения (source /cvmfs/.../setup.sh)
  5. Запуск вашей программы без интерактивного ввода
  6. Сохранение результатов с уникальным именем $(Cluster).$(Process)

## Файл №2: Описание задания (job.sub)

- **Что это такое:** текстовый файл с инструкциями для планировщика HTCondor — какие ресурсы нужны одной задаче и как её запускать.
- **universe = vanilla** — стандартный режим для пакетных задач.
- **executable и arguments** — путь к вашему скрипту и передаваемые ему параметры (например, $(ClusterId).$(ProcId)).
- **accounting_group = taiga** — учётная группа вашего проекта.
- **+SingularityImage = "/cvmfs/..."** — путь к образу контейнера в CVMFS, внутри которого запустится скрипт.
- **request_cpus, request_memory, request_disk** — сколько ядер CPU, оперативной памяти и места в scratch нужно ОДНОЙ задаче.
- **log, output, error** — файлы для журнала HTCondor, stdout и stderr.
- **queue N** — сколько независимых копий задачи поставить в очередь.

## Как нумеруются задачи: макросы $(ClusterId) / $(Cluster) и $(ProcId) / $(Process)

- **$(ClusterId) (или $(Cluster)):** уникальный номер всей вашей отправки (пачки задач), который присваивает condor_submit (например, 123456).
- **$(ProcId) (или $(Process)):** порядковый номер конкретной задачи внутри этой пачки от 0 до N-1 (например, при queue 10 будут задачи от 123456.0 до 123456.9). Всегда включайте $(Cluster).$(Process) в имена выходных файлов и логов, чтобы параллельные задачи не перезаписали друг друга!

# Два способа работы с файлами: когда использовать YES, а когда NO

Как правильно читать входные данные и сохранять результаты, чтобы не перегрузить сеть и не повредить файлы

## Сценарий А: Небольшие задачи и тесты

```
should_transfer_files = YES   (Копирование через HTCondor)
```

- **Как работает:** HTCondor сам копирует указанные файлы (transfer_input_files) с submit-узла в scratch рабочего узла, а по окончании возвращает результаты (transfer_output_files) обратно.
- **Когда удобно:** для малых тестовых задач, отладки и небольших текстовых результатов, когда файлы лежат только на submit-узле.
- **Опасность для больших данных:** весь трафик идёт через submit-узел! Например, 10 000 задач по 20 МБ создадут 200 ГБ сетевого трафика через входную машину.
- **Нюанс:** если файл из transfer_output_files не создался, задача уйдёт в ошибку (состояние Held).

## Сценарий Б: Основной режим NCP (Большие данные)

```
should_transfer_files = NO   (Работа напрямую с CephFS)
```

- **Рекомендовано сисадминами NCP:** общие каталоги CephFS (/taiga, /juno, /nova, /baikalgvd-nova) уже примонтированы на всех рабочих узлах по тем же путям!
- **Как работает:** HTCondor не гоняет файлы через submit-узел. Рабочий узел запускает скрипт прямо из CephFS, скрипт сам копирует нужный входной файл в свой локальный scratch и по завершении публикует результат в CephFS.
- **Почему не IF_NEEDED:** избегайте неоднозначного значения IF_NEEDED — всегда явно задавайте NO (для CephFS) или YES (для малых файлов).

## Три золотых правила безопасной работы с локальным диском (Scratch) и CephFS

- **1. Всегда переходите в $_CONDOR_SCRATCH_DIR в начале скрипта:** при should_transfer_files = NO задача стартует в общей папке initialdir, а не в scratch! Обязательно пишите в начале скрипта: cd "${_CONDOR_SCRATCH_DIR:?}" и создавайте все временные файлы только там.
- **2. Никогда не пишите временные файлы разных задач в одну общую папку CephFS:** параллельные задачи перезапишут промежуточные файлы друг друга, а очистка по маске (rm *.tmp) удалит чужие данные.
- **3. Атомарная публикация результата в CephFS:** сначала скопируйте готовый файл из scratch во временный файл на CephFS (через mktemp), а затем переименуйте его (mv) в финальное имя на том же диске CephFS. Такой mv атомарен — другие программы никогда не увидят недокопированный файл.

# Пример запуска задачи №1: старт с контейнером и возвратом файла

Готовый шаблон для знакомства с HTCondor и запуска программы в контейнере со сбросом результата на submit-узел

## Шаг 1. Создаём каталог и скрипт запуска run.sh

**1. Подготовка рабочей папки на submit-узле**

```
$ mkdir -p ~/htcondor-example/logs
$ cd ~/htcondor-example
$ chmod u+x run.sh    # после создания run.sh
```

**2. Содержимое файла run.sh**

```
#!/bin/bash
set -euo pipefail
job_id="$1"

# Замените этот блок командами своего софта
# (например: source setup.sh && ./my_program):
{
    printf 'Job: %s\n' "$job_id"
    hostname
    date -Is
} > "result-${job_id}.txt"
```

▸ **Что делает этот скрипт:** принимает идентификатор задачи (ClusterId.ProcId), записывает в выходной файл имя рабочего узла (hostname), на котором реально выполнился расчёт, и точное время.

▸ **Переход к своей программе:** укажите ваши бинарники и входные файлы в transfer_input_files (внутри задачи они будут лежать в текущем каталоге ./).

## Шаг 2. Файл описания задания job.sub (с контейнером CVMFS)

**Содержимое файла job.sub (группа taiga)**

```
universe = vanilla
executable = run.sh
arguments = $(ClusterId).$(ProcId)

accounting_group = taiga
# Опубликованный контейнер в CVMFS (пример с Geant4):
+SingularityImage = "/cvmfs/containers.jinr.ru/taiga/containers/taiga_g4:el9"
requirements = (HasSingularity == true)

request_cpus = 1
request_memory = 512MB
request_disk = 1GB

getenv = False
should_transfer_files = YES
when_to_transfer_output = ON_EXIT
# Дополнительные входные файлы (бинарник, конфиги):
# transfer_input_files = my_program, input.dat, setup.sh
transfer_output_files = result-$(ClusterId).$(ProcId).txt

log = logs/condor.$(ClusterId).log
output = logs/job.$(ClusterId).$(ProcId).out
error = logs/job.$(ClusterId).$(ProcId).err

queue $(jobs:1)
```

# Пример запуска задачи №2: работа с общим хранилищем CephFS и Scratch

Рекомендуемый шаблон для реальных расчётов и больших объёмов данных (should_transfer_files = NO)

## 1. Скрипт example_script.sh (CephFS → Scratch → CephFS)

/taiga/your_name/ncp-example/example_script.sh

```
#!/bin/bash
set -euo pipefail

shared_dir=$1; cluster=$2; process=$3
# Обязательно переходим в локальный scratch рабочего узла:
cd "${_CONDOR_SCRATCH_DIR:?HTCondor scratch is not set}"

# 1. Копируем входной файл только этой задачи из CephFS в scratch:
cp -- "$shared_dir/inputs/input-$process.dat" input.dat

# 2. Запускаем анализ/расчёт локально в scratch:
wc -l < input.dat > result.txt

# 3. Безопасно публикуем результат в CephFS (mktemp + атомарный mv):
final="$shared_dir/results/result-$cluster.$process.txt"
publish_tmp=$(mktemp "$shared_dir/results/.result-$cluster.$process.XXXXXX")
trap 'rm -f -- "$publish_tmp"' EXIT
cp -- result.txt "$publish_tmp"
mv -f -- "$publish_tmp" "$final"
```

▸ **Подготовка папок на CephFS перед запуском:** mkdir -p $shared_dir/{inputs,results,logs} && chmod +x $shared_dir/example_script.sh (положите входные файлы input-0.dat .. input-9.dat в inputs/).

## 2. Файл job.sub (без пересылки через submit-узел)

/taiga/your_name/ncp-example/job.sub

```
# Укажите учётную группу вашего эксперимента:
accounting_group = taiga
shared_dir = /taiga/your_name/ncp-example
universe = vanilla
# Раскомментируйте для запуска внутри контейнера CVMFS:
# +SingularityImage = "/cvmfs/<repo>/<path-to-image>"

executable = $(shared_dir)/example_script.sh
arguments = $(shared_dir) $(Cluster) $(Process)
initialdir = $(shared_dir)
should_transfer_files = NO

request_cpus = 1
request_memory = 1 GB
request_disk = 1 GB
# Разрешаем безопасный перезапуск при обслуживании узла:
+Evictable = true

log = $(shared_dir)/logs/job-$(Cluster).log
output = $(shared_dir)/logs/job-$(Cluster).$(Process).out
error = $(shared_dir)/logs/job-$(Cluster).$(Process).err
queue 10
```

▸ **Почему этот шаблон идеален для науки:** при падении узла и перезапуске задача начнёт счёт в чистом scratch и заменит финальный файл только после полного успеха.

# Команды отправки, контроля очереди и диагностики проблем

Как проверить синтаксис до запуска, отследить статус расчёта, проверить код возврата и разобраться с ошибками

## 1. Проверка описания и отправка в очередь

**Отправка тестовой задачи и серии**

```
# Сухая проверка описания без реального запуска:
$ condor_submit -dry-run logs/preview.ads job.sub
# Отправка 1 пробного задания (всегда начинайте с 1 задачи!):
$ condor_submit job.sub
# Отправка серии из 10 независимых заданий (ProcId 0..9):
$ condor_submit job.sub jobs=10
```

▸ **Важно:** в ответе condor_submit выведет номер кластера (например, submitted to cluster 123456). Сохраните его для проверок!

## 2. Мониторинг очереди и расшифровка статусов

**Просмотр активных задач в очереди**

```
$ condor_q                    # сводка по вашим задачам
$ condor_q 123456 -nobatch    # подробно по каждой задаче
$ condor_watch_q              # живой мониторинг в реальном времени
```

**I (Idle)**
Ждёт свободных ресурсов в очереди

**R (Running)**
Выполняется на рабочем узле

**H (Held)**
Ошибка! Задача приостановлена

## 3. Проверка завершения (выход из очереди — ещё не успех!)

**Проверка кода возврата и памяти через condor_history**

```
# Проверяем статус завершённых задач:
$ condor_history 123456 -af ClusterId ProcId JobStatus ExitCode ExitBySignal
# Узнаём реальное пиковое потребление памяти (в МБ):
$ condor_history 123456 -af MemoryUsage
```

▸ **Признак успешного расчёта:** JobStatus = 4 (Completed), ExitCode = 0, ExitBySignal = false + проверка логов в logs/*.err и непустых файлов результата.

## 4. Диагностика сбоев (Held / долгий Idle) и отмена задач

**Разбор проблем и снятие задач с очереди**

```
# Узнать точную причину остановки задачи в состоянии Held (H):
$ condor_q 123456 -hold
# Почему задача долго ждёт в состоянии Idle (I):
$ condor_q 123456 -better-analyze
# Удалить одну задачу (123456.0) или всю отправку целиком (123456):
$ condor_rm 123456.0    # или: condor_rm 123456
```

▸ **Частые причины Held:** не создан файл из transfer_output_files, опечатка в пути к контейнеру +SingularityImage или скрипт без chmod +x.

# Шесть важных правил выбора ресурсов и культуры работы на NCP

Рекомендации системных администраторов платформы: как избежать «голодания задач» и считать максимально быстро

### 1. Выбирайте 1 ядро (request_cpus = 1)

Разбивайте расчёты на независимые одноядерные задачи! Запрос request_cpus = 8 требует 8 свободных ядер на ОДНОМ узле (4+4 на разных узлах объединить нельзя). На загруженном кластере многоядерные задачи могут ждать в очереди бесконечно (job starvation).

### 2. Точно запрашивайте память и диск

На узлах доступно от 4 до 14 ГБ ОЗУ и ~20 ГБ scratch на ядро. Замеряйте реальный расход на тестовом запуске (condor_history <id> -af MemoryUsage) и берите разумный запас. Завышенный request_memory резервирует память в слоте и мешает запуску других задач!

### 3. Всегда указывайте accounting_group

В каждом job.sub указывайте свою группу: taiga, baikalgvd, juno, nova, dune, borexino, dayabay, darkside, gna (без неё счёт уйдёт в common). Для JUNO учёт accounting_group = juno критически важен для официальной отчётности вклада ОИЯИ!

### 4. Оптимальное время задачи — до 24 часов

Не дробите счёт на секундные задачи (накладные расходы на старт), но старайтесь укладываться в < 24 ч. Для коротких задач полезно добавить подсказку +EstimatedRuntimeSeconds = 3600. О задачах длительностью в несколько суток заранее предупреждайте админов в Telegram.

### 5. Идемпотентность и +Evictable = true

Из-за сбоев железа HTCondor может перезапустить задачу с начала на другом узле. Скрипт должен работать без ручной очистки старых файлов. Если задачу безопасно перезапускать с нуля, добавьте в job.sub строку +Evictable = true.

### 6. Разные Seeds для генераторов случайных чисел

При каждом новом вызове condor_submit нумерация $(ProcId) снова начинается с 0! Если использовать только $(ProcId) как seed в Монте-Карло (Geant4/CORSIKA), разные отправки сгенерируют одинаковые события. Комбинируйте $(ClusterId) и $(ProcId) или передавайте таблицу сидов.

# Чек-лист перед запуском расчёта и полезные ссылки

Краткая памятка для самопроверки перед отправкой больших серий задач и контакты поддержки

## Чек-лист пользователя перед отправкой серии задач

- **1. Вы НЕ запускаете расчёт напрямую в терминале:** на submit-узле (10.220.18.43) выполняется только сборка и вызов condor_submit.
- **2. Проверено членство в Linux-группе:** команда id -nG показывает группу вашего проекта (например, taiga), а в job.sub указан соответствующий accounting_group.
- **3. Вы вышли из контейнера сборки:** после компиляции программы внутри apptainer exec вы выполнили exit и запускаете condor_submit из обычного shell.
- **4. Временные файлы пишутся в Scratch:** скрипт переходит в cd "$_CONDOR_SCRATCH_DIR", а для больших данных используется CephFS с should_transfer_files = NO.
- **5. Выполнена проверка на 1 тестовой задаче:** сначала запущен condor_submit -dry-run и 1 пробный расчёт; проверены логи logs/*.err, *.out и реальный MemoryUsage.
- **6. Уникальные имена файлов и разные Seeds:** выходные файлы содержат $(Cluster).$(Process), а генераторы случайных чисел не дублируют сиды между отправками.

## Официальные инструкции и документация

- **Гайд системных администраторов NCP (Bootcamp Wiki):**
  `git.jinr.ru/neutrino_group/bootcamp/-/wikis/Neutrino Computing Platform`
- **Гайд группы TAIGA по запуску задач в HTCondor:**
  `git.jinr.ru/taiga/common/-/wikis/Запуск-задач-на-нейтринной-платформе-через-HTCondor`
- **Регистрация в CICC ОИЯИ и электронная анкета:**
  `lxs-s03.jinr.ru/cicc/index.php/registration_at_cicc-ru/`
- **Публикация контейнеров в CVMFS:**
  `git.jinr.ru/cvmfs-container-sync/cvmfs-singularity-sync`

## Контакты для получения доступа и техподдержки

- **Оформление аккаунтов и групп (ЛИТ ОИЯИ):** Наталья Громова (grom@jinr.ru)
- **Руководитель темы (02-2-1099-3-2015/2026):** Бородин А.Н. (borodin@jinr.ru)
- **Координатор по доступу в группе TAIGA:** Александр Блинов (Telegram: @Alspy)
- **Чат техподдержки пользователей NCP в Telegram:** Computer Support Group (t.me/+OneWb_SPxv45MzBh) — при сбоях узлов указывайте hostname, Job ID и текст ошибки.