🤖 Параллельная разработка с Claude Code и Git Worktree

🚀 Параллельная разработка с несколькими инстансами Claude Code

Ключевые тезисы:

- ✅ Можно запускать несколько окон Claude Code для одновременной работы над разными частями проекта
- ❌ Прямой запуск приводит к конфликтам файлов между агентами
- 🔧 Решение — использование **Git Worktree** для изоляции копий проекта
- 🎯 Каждый инстанс работает в своей изолированной копии, а затем изменения объединяются

🎯 Проблема параллельной работы

При одновременном запуске нескольких инстансов Claude Code (например, для фронтенда, бэкенда, БД и дизайна) возникает проблема:

> ***Конфликты перезаписи файлов** — разные агенты изменяют одни и те же файлы, что ломает систему и снижает эффективность.*

Пример: один агент создает `app_backend.js`, а другой, отвечающий за SQL, также изменяет этот файл.

💡 Решение: Git Worktree

Worktree — это технология Git, позволяющая создавать **изолированные копии проекта** в разных директориях.

Как это работает:

- Каждому инстансу Claude Code дается команда использовать skill `worktree`
- Создается отдельная копия всего проекта для каждого агента
- Агенты работают в своих изолированных средах
- После завершения работы изменения объединяются через `git merge`

Пример структуры после создания worktree:

```
/project
├── .claude/
│   └── worktrees/
│       ├── backend/      # Копия для бэкенд-разработчика
│       ├── frontend/     # Копия для фронтенд-разработчика
│       └── design/       # Копия для дизайнера
└── (основные файлы проекта)
```

📚 Основные концепции Git

Git — система контроля версий, отслеживающая изменения в проекте.

Ключевые термины:

- **Репозиторий** — папка со скрытой директорией `.git`, хранящей историю проекта
- **Коммит (commit)** — точка сохранения состояния проекта (как чекпоинт в игре)
- **Ветка (branch)** — отдельная линия разработки с собственной историей коммитов
- **Слияние (merge)** — объединение изменений из разных веток
- **Конфликт** — ситуация, когда несколько веток изменяют одни и те же участки кода

Пример workflow:

1. Работа над основным UI → коммиты в ветке `main`
2. Срочная задача (добавить тумблер) → создаем ветку `hotfix`
3. Параллельно продолжаем работу над графиками в `main`
4. После завершения — объединяем ветки через `merge`

🛠️ Практическое применение

Процесс работы с несколькими инстансами:

1. **Создание общего плана**

```
"Создай план для проекта 'Крестики-нолики в стиле аниме':
- Бэкенд (логика, API)
- База данных (пользователи, рейтинг)
- Фронтенд (интерфейс)
- Дизайн (стили, анимации)"
```

2. **Запуск изолированных агентов**

```
Агенту 1: "Используй skill worktree, работай в своей директории как фронтенд
Агенту 2: "Используй skill worktree, работай над дизайном"
Агенту 3: "Используй skill worktree, отвечай за бэкенд и SQL"
```

3. **Объединение результатов**

```
"Сделай merge всех веток из worktrees в основной проект.
Разреши конфликты, если они возникнут."
```

⚠️ Важные нюансы

Когда НЕ нужны worktree:

- При **read-only задачах** (только чтение, без изменения файлов)
- Когда нет риска конфликтов (агенты работают с совершенно разными файлами)

Альтернативные подходы:

- **Встроенная команда** `/branch` — создает субагентов с worktree автоматически
- **Agent Teams** — команда агентов с главным управляющим (дороже, но более интегрировано)

Преимущества ручного управления инстансами:

- Больше контроля над каждым агентом
- Возможность вмешательства в любой момент
- Работа не в терминале (в отличие от Agent Teams)

🔄 Доработка и фиксация проблем

Если после объединения остаются проблемы:

1. **Создание лога изменений**

```
"Создай файл log.txt с описанием всех внесенных изменений"
```

2. **Параллельный фикс разных частей**

```
Агенту 1: "Создай worktree 'backend-fix', исправь проблемы с регистрацией"
Агенту 2: "Создай worktree 'frontend-optimize', улучши производительность и д
```

3. **Интеграция и тестирование**

```
"Создай worktree для стыковки бэкенда и фронтенда.
Протестируй через Playwright MCP и разверни через Docker."
```

Резюме: Ключевые выводы

1. **Git Worktree** — мощный инструмент для **параллельной разработки** без конфликтов
2. Каждый агент работает в **изолированной копии проекта** со своей веткой
3. **Частые коммиты** в каждой ветке позволяют откатываться при проблемах
4. **Объединение через merge** — финальный этап, где Claude может авторазрешать конфликты
5. **GitHub** служит для бэкапа и хранения истории (важно: `.env` в `.gitignore`, приватные репозитории)

Итог: Использование нескольких инстансов Claude Code с worktree ускоряет разработку в 4-5 раз, сохраняя контроль и избегая конфликтов.
