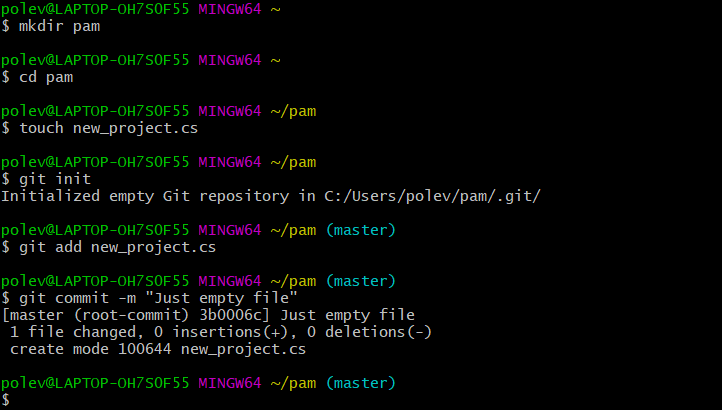
7 Вариант: Использовать в работе 2 разных файла, для одного из них использовать 3 различные ветки.

**Создание проекта и репозитория**

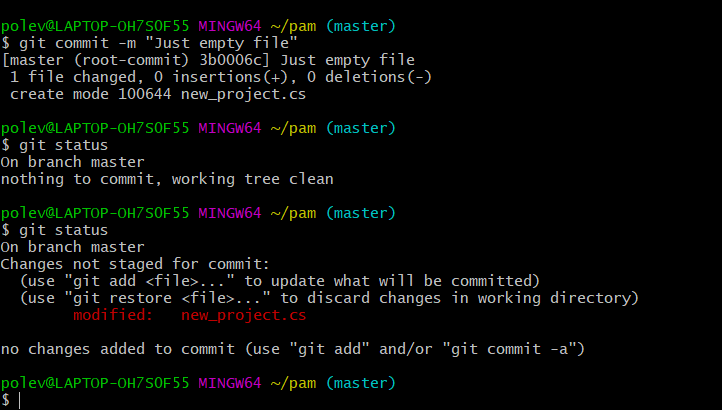


**Проверка состояния репозитория**

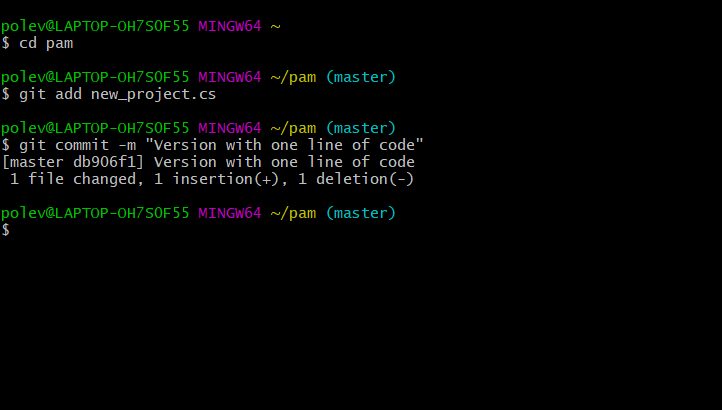


**Индексация изменений**

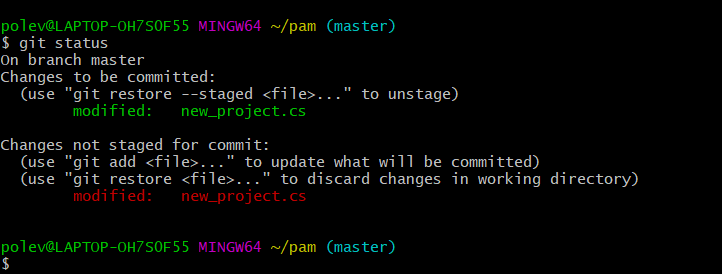
После внесения изменений в файл проекта проверяю состояние репозитория. Репозиторий отследил изменения:



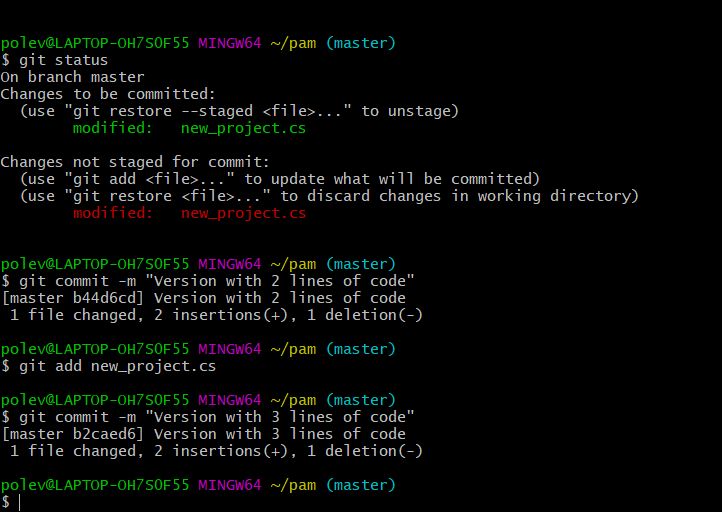
Идексирую изменения и добавляю коммит:



Вновь вношу изменения в файл проекта, индексирую изменения, опять изменяю файл и проверяю состояния репозитория. Можно заметить, что система различает версии одного и того же файла:

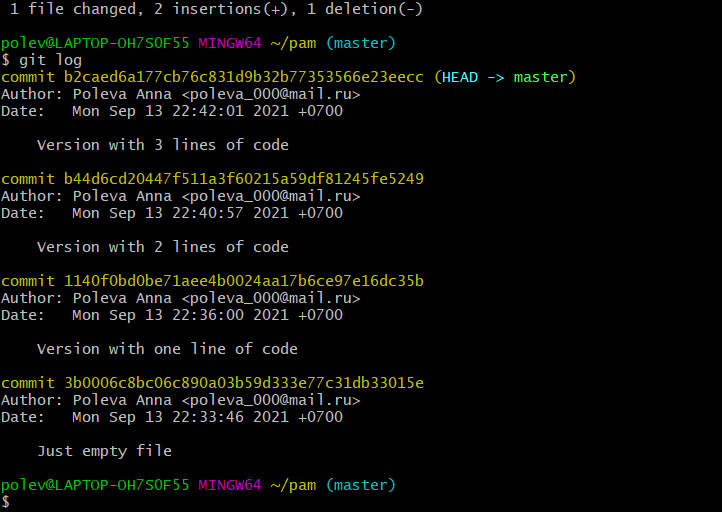


Добавляю коммит к проиндексированной версии. Индексирую последнюю версию и добавляю коммит к ней:

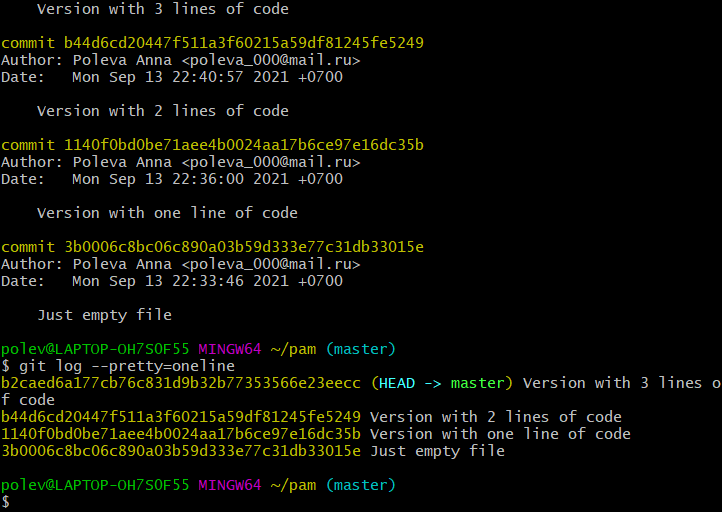


**История версий**

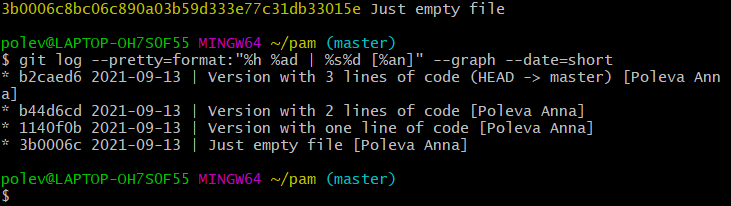
Для того, чтобы посмотреть всю историю версий выполняю команду *git log*

**

Для удобства восприятия вывожу иторию в режиме «одной строки»

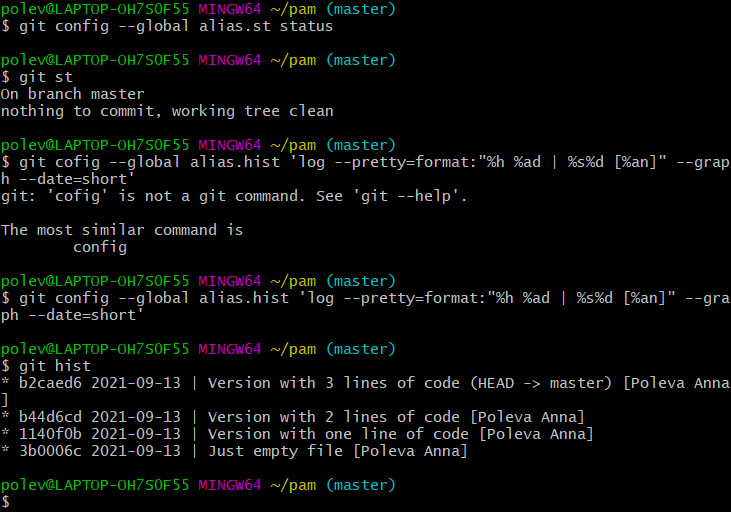


Для более удобного вывода использую следующий способ форматирования:



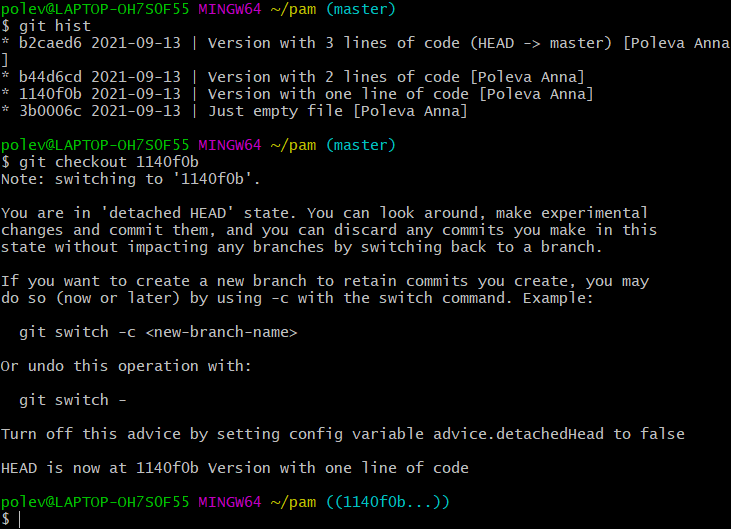
**Создание алиасов**

Чтобы увеличить скорость набора команд использую алиасы:

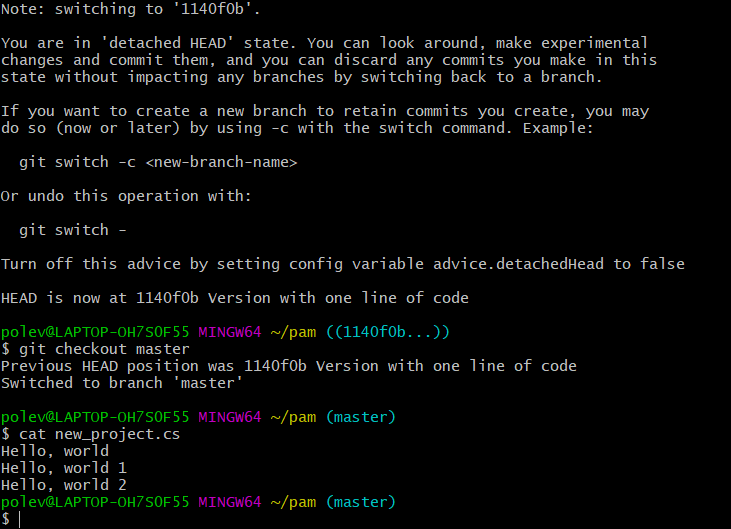


**Перемещение по версиям**

Возвращаюсь к прошлой версии файла с помощью команды *git checkout <hast>*

**

Для возврата к последней версии использую аналогичную команду:



**Тегирование**

Задаю тег для последней версии файла с помощью команды *git tag <some\_tag>*:

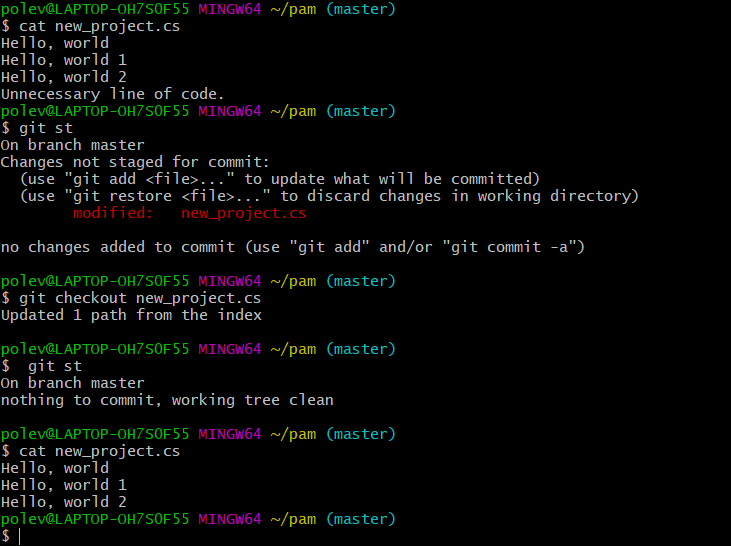


**Клонирование репозиториев**

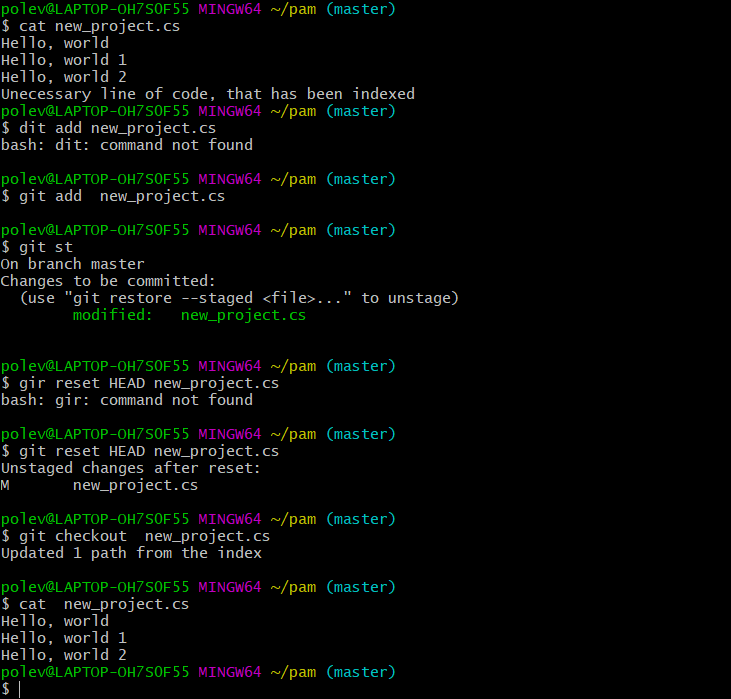
Для клонирования репозитория использую команду *git clone <url>*

**Отмена ошибочных изменений, коммитов**

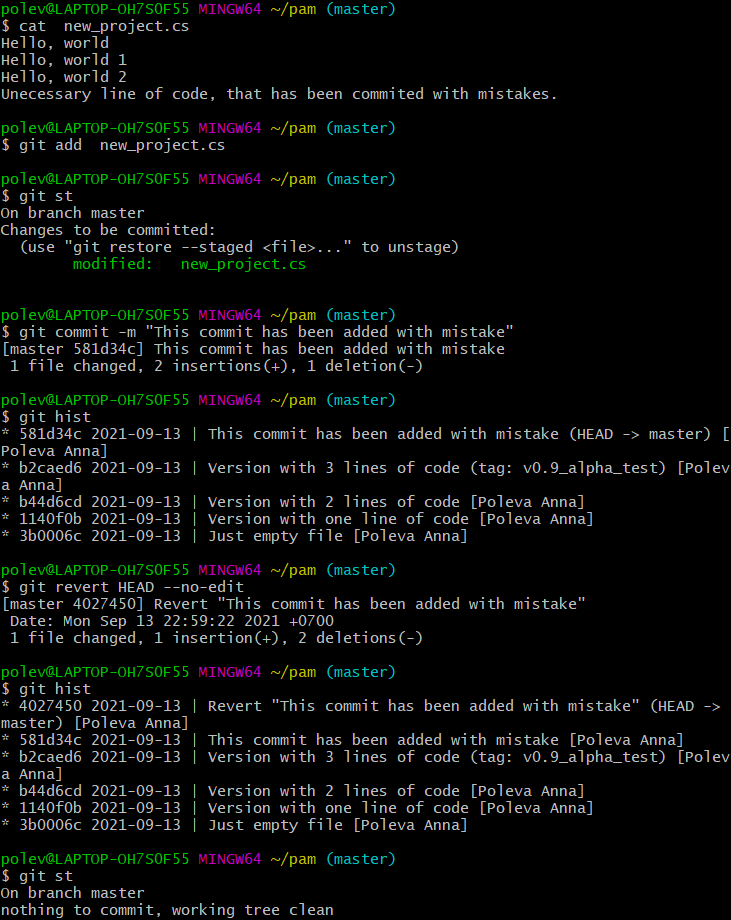
Добавляю в файл ошибочную строку. Для того чтобы вернуться к последней проиндексированной версии использую команду *git checkout <file’s\_name>*:



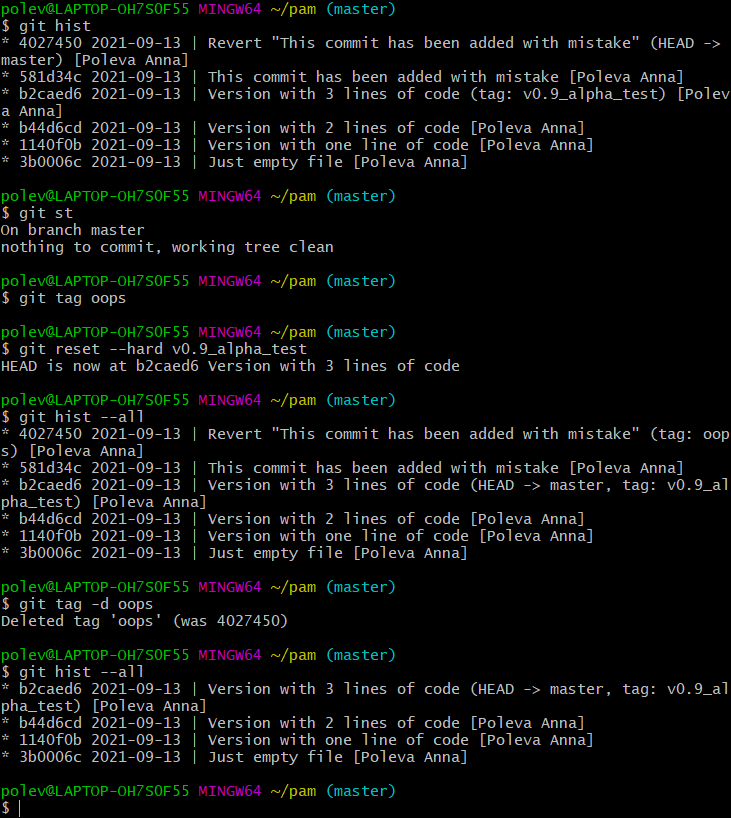
Для того чтобы исправить проиндексированные ошибки использую команду *git reset HEAD <file’s\_name>*:



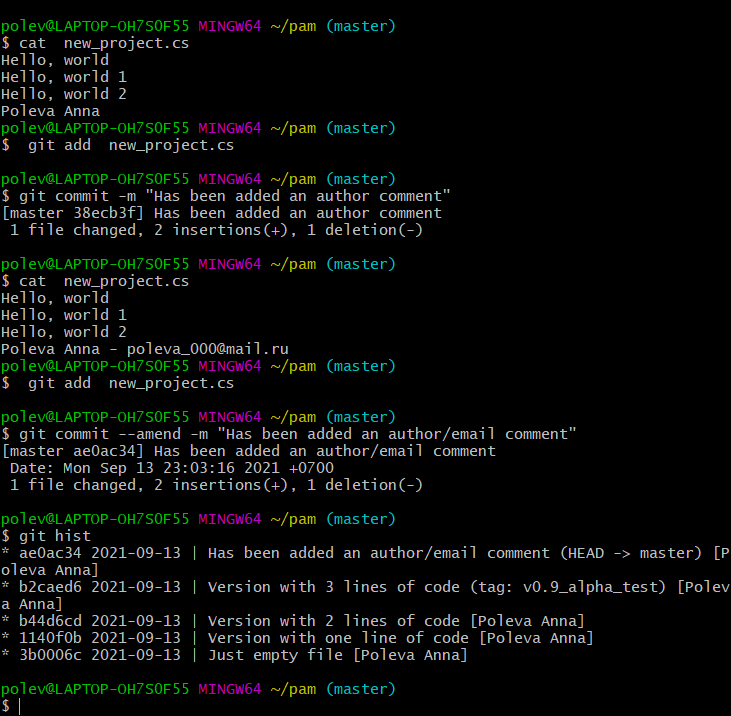
Для того чтобы исправить проиндексированные и добавленные в коммит ошибки ошибки использую команду *git revert HEAD*:



Для удаления отмененного коммита из истории тегирую текущую версию файла. После этого выполняю сброс к предыдущей версии.

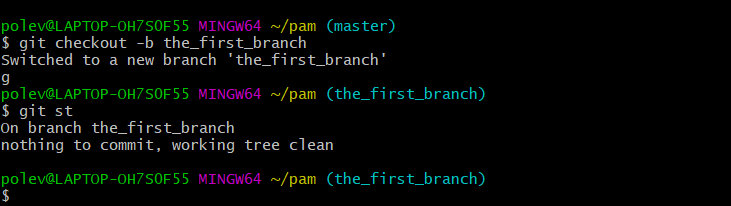


Существует возможность исправления коммита без его отмены и удаления:

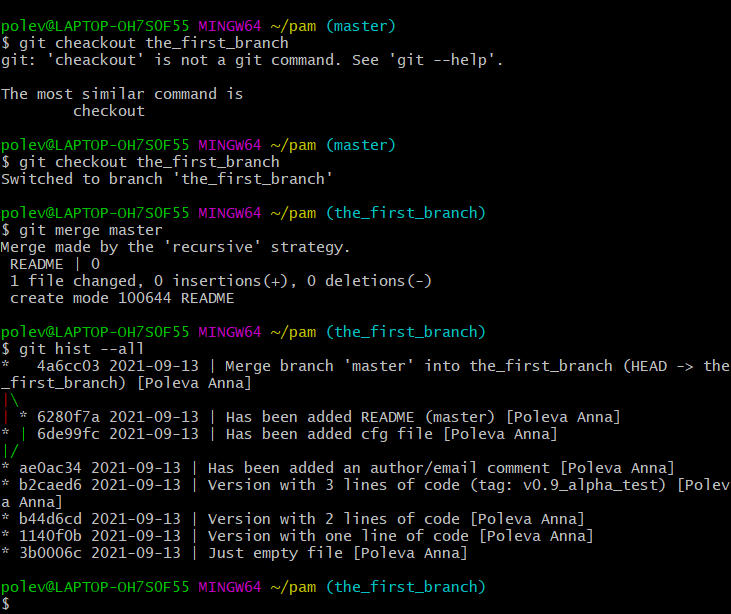


**Ветвление и разрешение конфликтов при ветвлении**

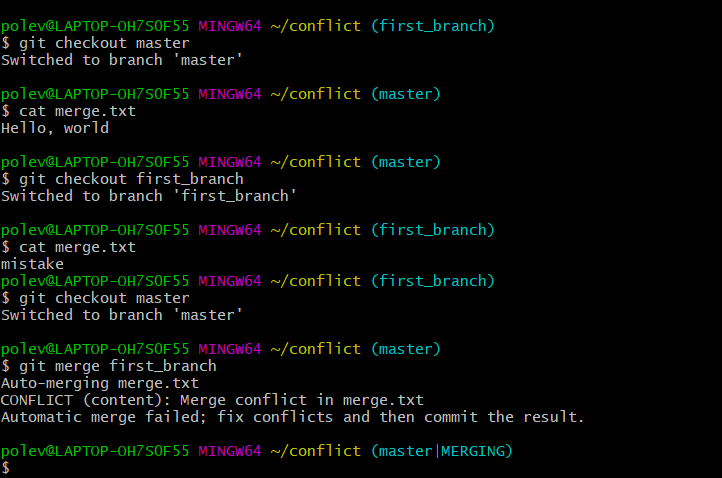
Создание ветки:



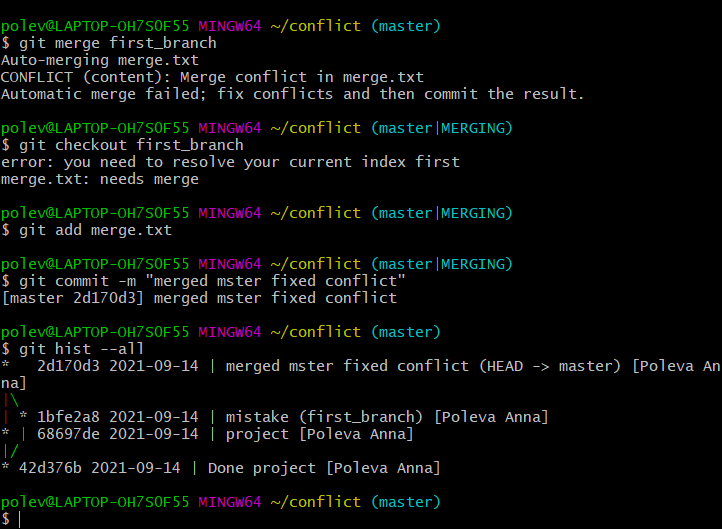
Слияние двух веток:



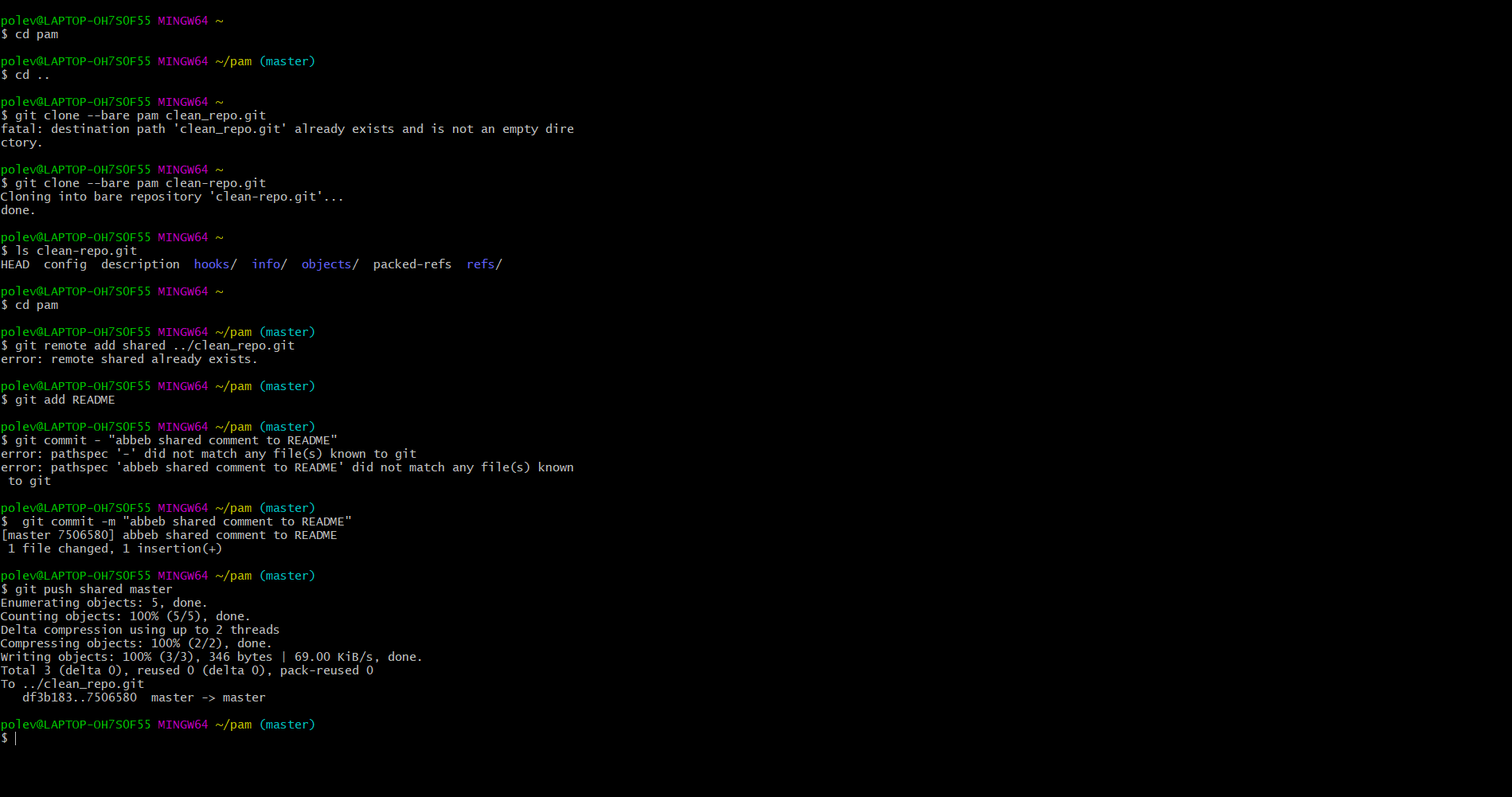
Создание конфликта:



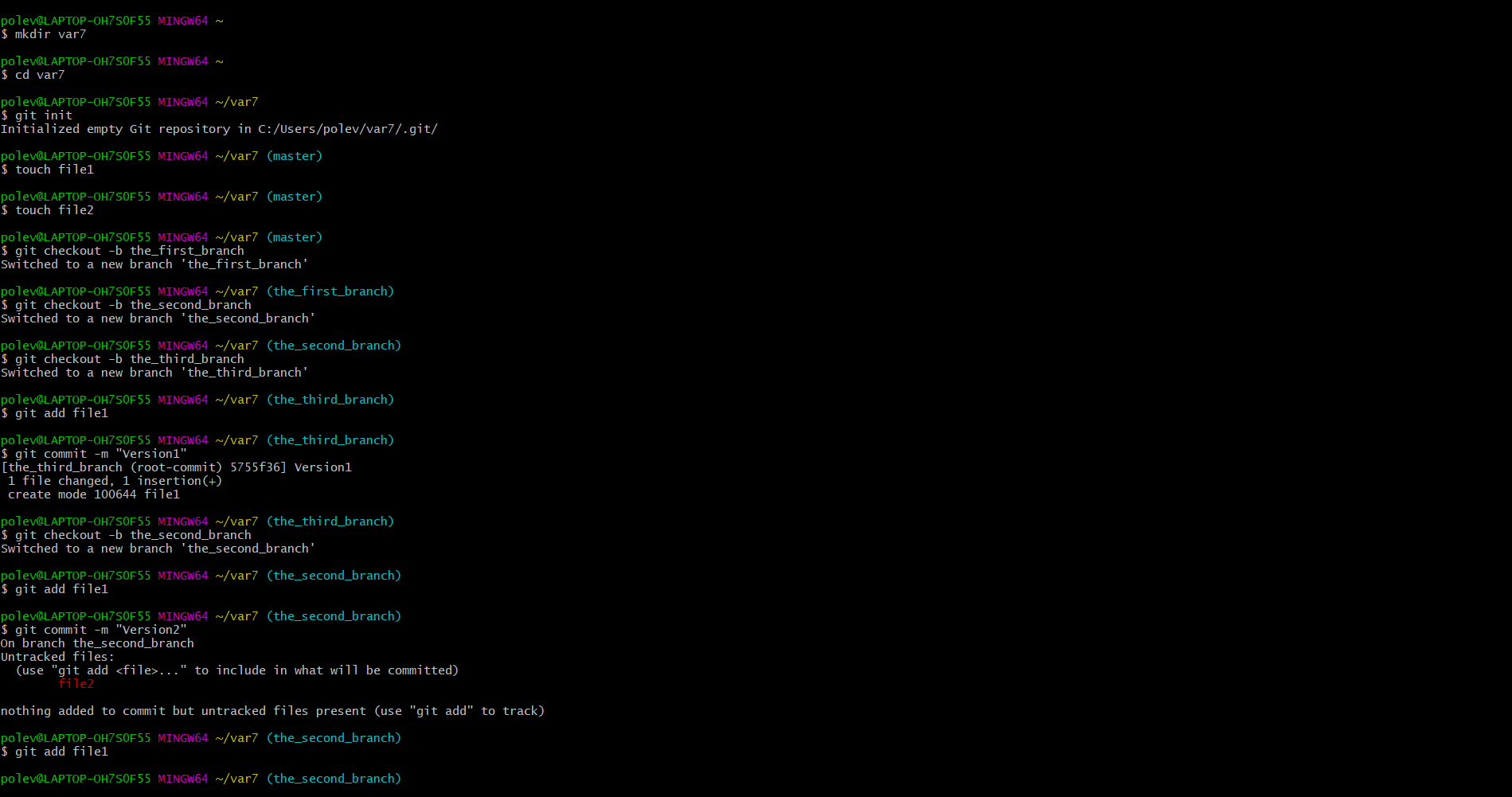
Разрешение конфликта:

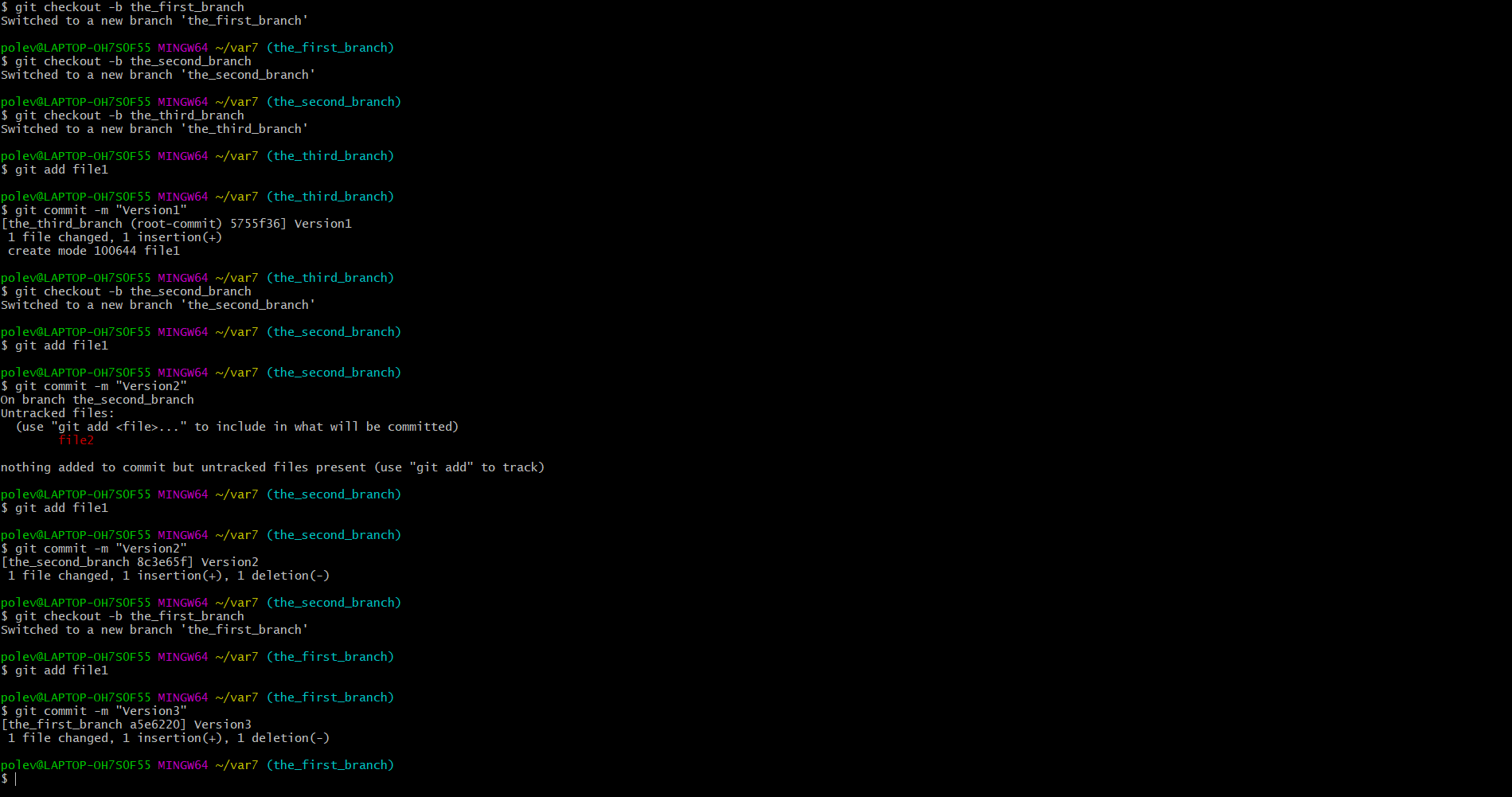


**Создание чистого репозитория**

****

**7 Вариант: Использовать в работе 2 разных файла, для одного из них использовать 3 различные ветки.**

****

****

**Контрольные вопросы**

1. **Что представляет собой система GIT?**

Git — это система контроля версий, которая помогает отслеживать историю изменений в файлах.

1. **Перечислите возможности систем управления версиями.**

Позволяет отслеживать изменения в документах, при необходимости производить их откат, определять, кто и когда внес исправления и т.п

1. **Что такое коммит и когда он выполняется?**

Коммит – фиксация изменений, внесенных в индекс. Эта операция берет все подготовленные изменения и отправляет их в репозиторий как единое целое.  Коммиты разбивают процесс разработки, состоящий из большого количества правок, на отдельные шаги.

1. **Как посмотреть историю коммитов?**

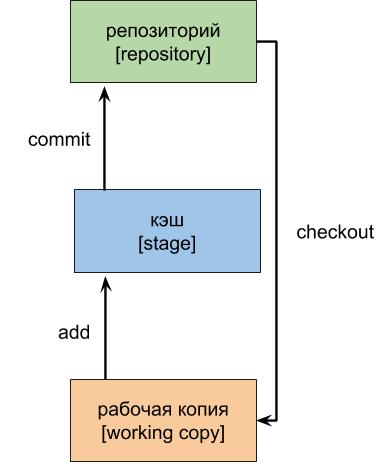
Чтобы посмотреть историю коммитов нужно воспользоваться командой *git log*

1. **Как создать пустой GIT репозиторий?**

Для того чтобы создать репозиторий, для начала, нужно создать папку, в которой он будет располагаться с помощью команды *mkdir «name»*. Теперь нужно перейти в каталог с помощью команды *cd «name»* и наконец создать в нем пустой git репозиторий, воспользовавшись командой *git init*.

1. **Какую архитектуру имеет система контроля версий GIT?**

Система контроля версий *git*имеет архитектуру трех деревьев. Схематично она выглядит так как показано на рисунке ниже.



1. **Как создать новую ветку в Git соединить ветки?**

Чтобы создать ветку и сразу переключиться на нее, можно выполнить команду *git checkout* с параметром *–b*.  
Для объединения двух веток проще всего перейти на целевую ветку (например *master*) с помощью команды *git checkout master* и выполнить команду слияния *git merge some\_branch* где *some\_branch* - это ветка, которую мы хотим влить в *master*.

1. **Поясните понятие «staging area».**

Область подготовленных файлов (**staging area**) — это обычный файл, обычно хранящийся в каталоге Git, который содержит информацию о том, что должно войти в следующий коммит. Иногда его называют индексом (index), но в последнее время становится стандартом называть его областью подготовленных файлов. Область подготовленных файлов это уже не рабочий каталог, но ещё и не коммит.

1. **Как сравнить версии файла в GIT?**

Для вывода изменений в файлах по сравнению с последним коммитом, используется *git diff* без параметров. Команда выводит изменения в файлах, которые еще не были добавлены в индекс. Сравнение происходит с последним коммитом.