

Как пользоваться Git для начинающих

[Инструкции](#) Я н в а р ь 24, 2017 [7admin](#)

Git — это очень популярная система контроля версий и совместной разработки проектов с открытым исходным кодом. С помощью Git вы можете отслеживать изменения в исходном коде своих проектов, возвращать предыдущие версии в случае критических ошибок, а также делиться своим кодом со всеми желающими и принимать от них исправления.

Это мощная система, которая позволяет оптимизировать работу над вашими проектами. Здесь нет каких-либо требований к языку или структуре файлов, поэтому у разработчиков полная свобода действий. В этой статье мы рассмотрим как пользоваться git для начинающих пользователей. Рассмотрим все очень подробно, начиная от настройки, и до ветвей проектов.

Содержание статьи:

- [Команда git](#)
- [Как работает git?](#)
- [Как пользоваться Git?](#)
 - [Создание проекта](#)
 - [Настройка проекта в git](#)
 - [Фиксация изменений](#)
 - [Отправка изменений](#)
 - [Управление ветвями](#)
- [Выводы](#)

К о м а н д а g i t

Уже по традиции, перед тем, как перейти к примерам и работе с командой давайте рассмотрим ее основные опции и параметры. Синтаксис git очень прост:

\$ gitопциикомандааргументы

Сначала рассмотрим опции, они влияют на работу всей утилиты:

- **-C** — использовать указанную папку репозитория вместо текущей папки;
- **-спараметр=значение** — использовать указанное значение параметра конфигурации;
- **-p** — прокручивать весь вывод с помощью less;

Теперь рассмотрим команды `git`, их немного больше и именно с помощью них вы будете выполнять все основные действия:

- **add** — добавить файл или папку в репозиторий `git`;
- **am** — применить все патчи из email;
- **archive** — создать архив файлов;
- **bisect** — использовать бинарный поиск для поиска нужного коммита;
- **branch** — управление ветками проекта;
- **bundle** — перемещение объектов и ссылок в архиве;
- **checkout** — переключение между ветками;
- **cherry-pick** — внести изменения в уже существующие коммиты;
- **clean** — удалить все неотслеживаемые файлы и папки проекта;
- **clone** — создать копию удаленного репозитория в папку;
- **commit** — сохранить изменения в репозиторий;
- **diff** — посмотреть изменения между коммитами;
- **fetch** — скачать удаленный репозиторий;
- **init** — создать репозиторий;
- **merge** — объединить две ветви;
- **pull** — интегрировать удаленный репозиторий с локальным;
- **push** — отправить изменения в удаленный репозиторий;
- **tag** — управление тегами;
- **worktree** — управление деревьями разработки.

Аргументы зависят от используемой команды, поэтому более подробно мы будем разбирать их в примерах.

Как работает git?

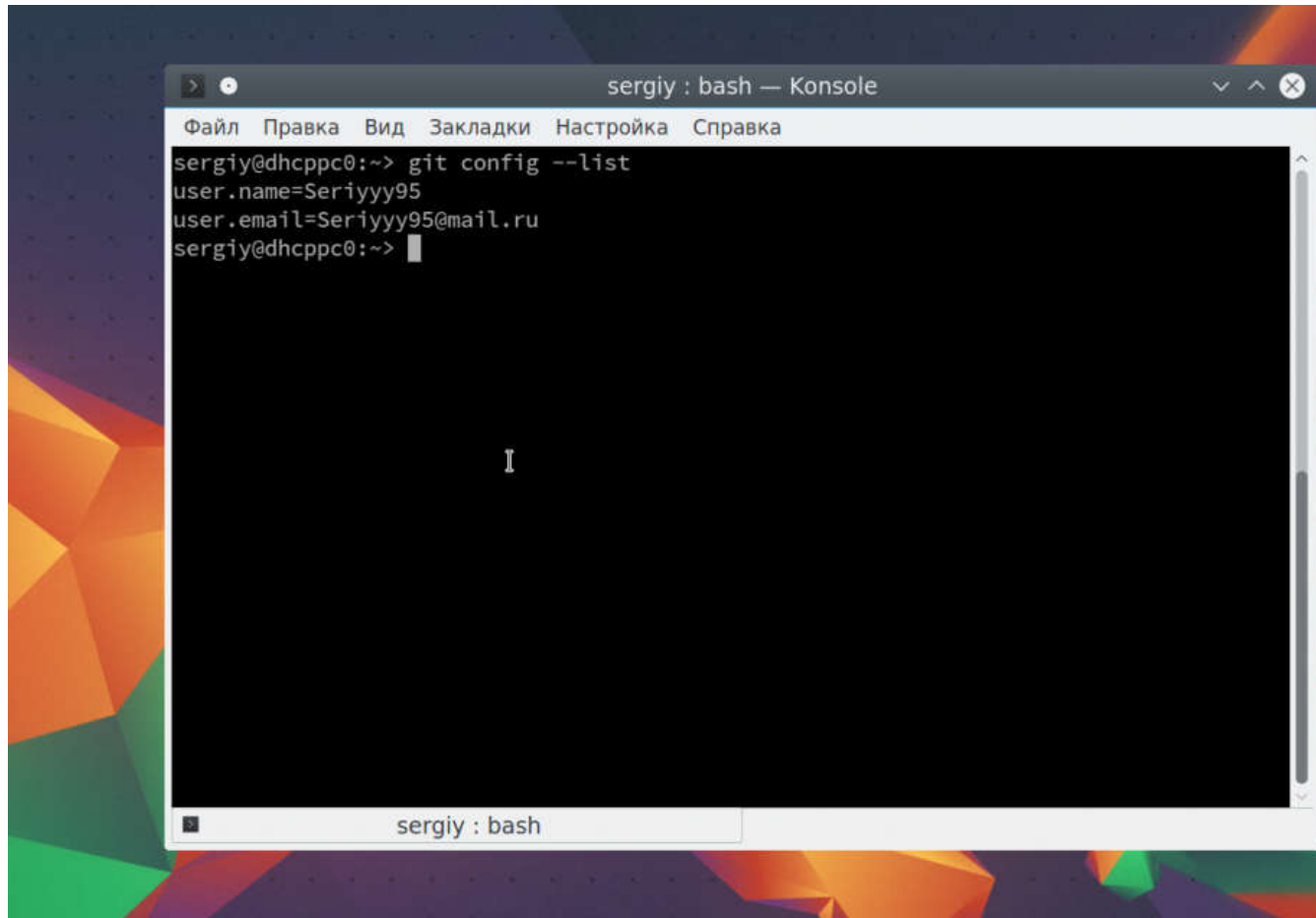
Перед тем как идти дальше и рассматривать использование `git` для управления своими проектами, я бы хотел сказать несколько слов о том, как же работает эта технология, так сказать, основы работы `git`.

Итак, из всего выше перечисленного, вы, наверное, уже поняли, что контроль версий позволяет вам посмотреть изменения на любом этапе разработки, а также вернуться к любому моменту. Но это не совсем так. Изменения сохраняются в виде коммитов. По-русски — фиксация. Вы делаете начальный коммит, чтобы сохранить начальное состояние проекта, а затем для каждого изменения. Это работает как снимки состояния.

Кроме того, `git` позволяет отправлять данные на удаленный сервер. Отправляются не только готовая версия, но и все снимки, таким образом, любой человек из команды может посмотреть историю изменений. К каждому снимку нужно делать комментарий, так работа с `git` будет проще и понятнее.

Как пользоваться Git?

Дальше я буду предполагать, что вы выполнили установку и базовую настройку git. Кроме установки, вам нужно указать правильный адрес электронной почты и имя пользователя для доступа к серверу Git, например, на GitHub. Если вы этого еще не сделали смотрите инструкцию [установка Git в Ubuntu 16.04](#).

A screenshot of a terminal window titled 'sergiy : bash — Konsole'. The window has a menu bar with 'Файл', 'Правка', 'Вид', 'Закладки', 'Настройка', and 'Справка'. The terminal shows the command 'git config --list' being executed, resulting in two lines of output: 'user.name=Seriy95' and 'user.email=Seriy95@mail.ru'. The prompt 'sergiy@dhcppc0:~>' is visible at the bottom of the terminal area.

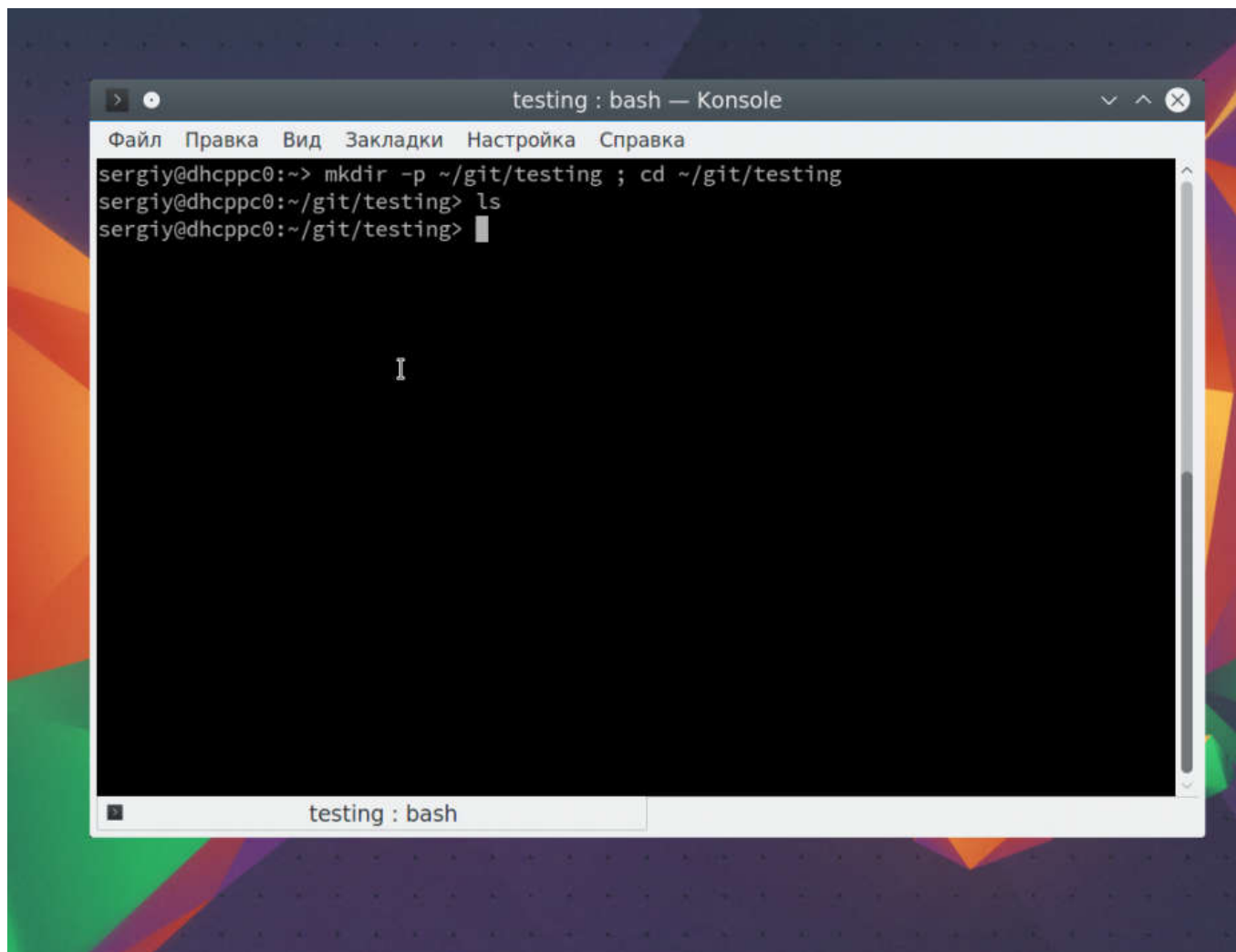
```
sergiy@dhcppc0:~> git config --list
user.name=Seriy95
user.email=Seriy95@mail.ru
sergiy@dhcppc0:~>
```

Обычно, структура проекта в Git будет зависеть от масштаба и сложности вашей программы. Но для начала мы будем использовать проект, состоящий только из одной ветки. Каждый проект содержит одну ветку по умолчанию, она называется master. Наш первый проект будет называться test.

С о з д а н и е п р о е к т а

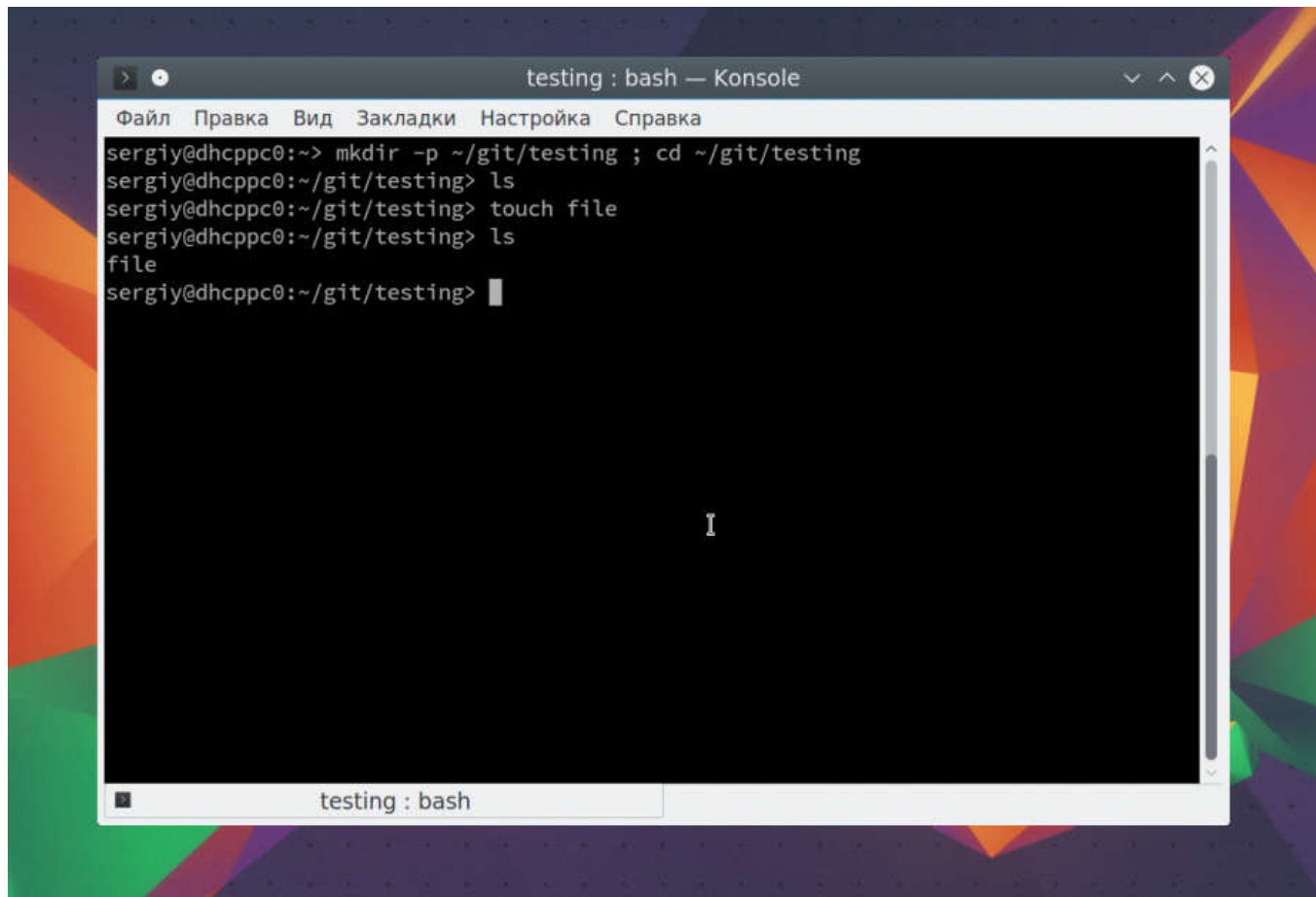
Когда настройка git завершена перейдем к вашему проекту. В самом начале вам достаточно создать папку для файлов проекта. Если вы собираетесь работать над несколькими проектами, создайте папку git в вашем домашнем каталоге, а уже туда поместите папки ваших проектов:

```
mkdir -p ~/git/testing ; cd ~/git/testing
```



Эта команда создаст нужную структуру папок и переводит текущий каталог в только что созданный. Теперь создадим первый файл нашего проекта:

```
touch file
```



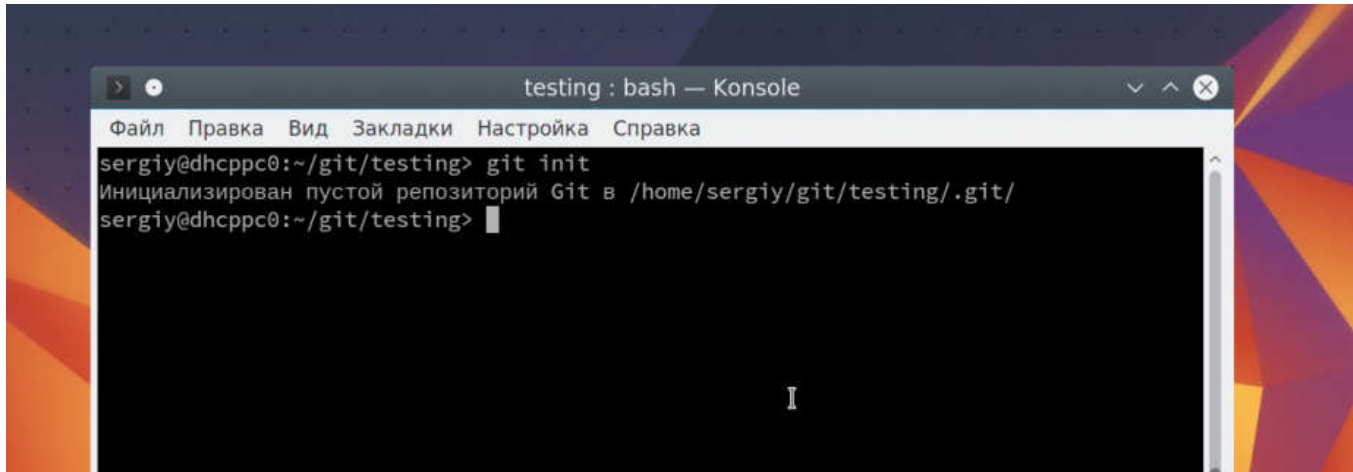
```
testing : bash — Konsole
Файл  Правка  Вид  Закладки  Настройка  Справка
sergiy@dhcppc0:~> mkdir -p ~/git/testing ; cd ~/git/testing
sergiy@dhcppc0:~/git/testing> ls
sergiy@dhcppc0:~/git/testing> touch file
sergiy@dhcppc0:~/git/testing> ls
file
sergiy@dhcppc0:~/git/testing> 
```

Проект готов, но система контроля версий git еще не знает об этом.

Н а с т р о й к а п р о е к т а в g i t

Перед тем как git начнет отслеживать изменения, нужно подготовить все необходимые конфигурационные файлы. Сначала инициализируем пустой репозиторий в нашей папке:

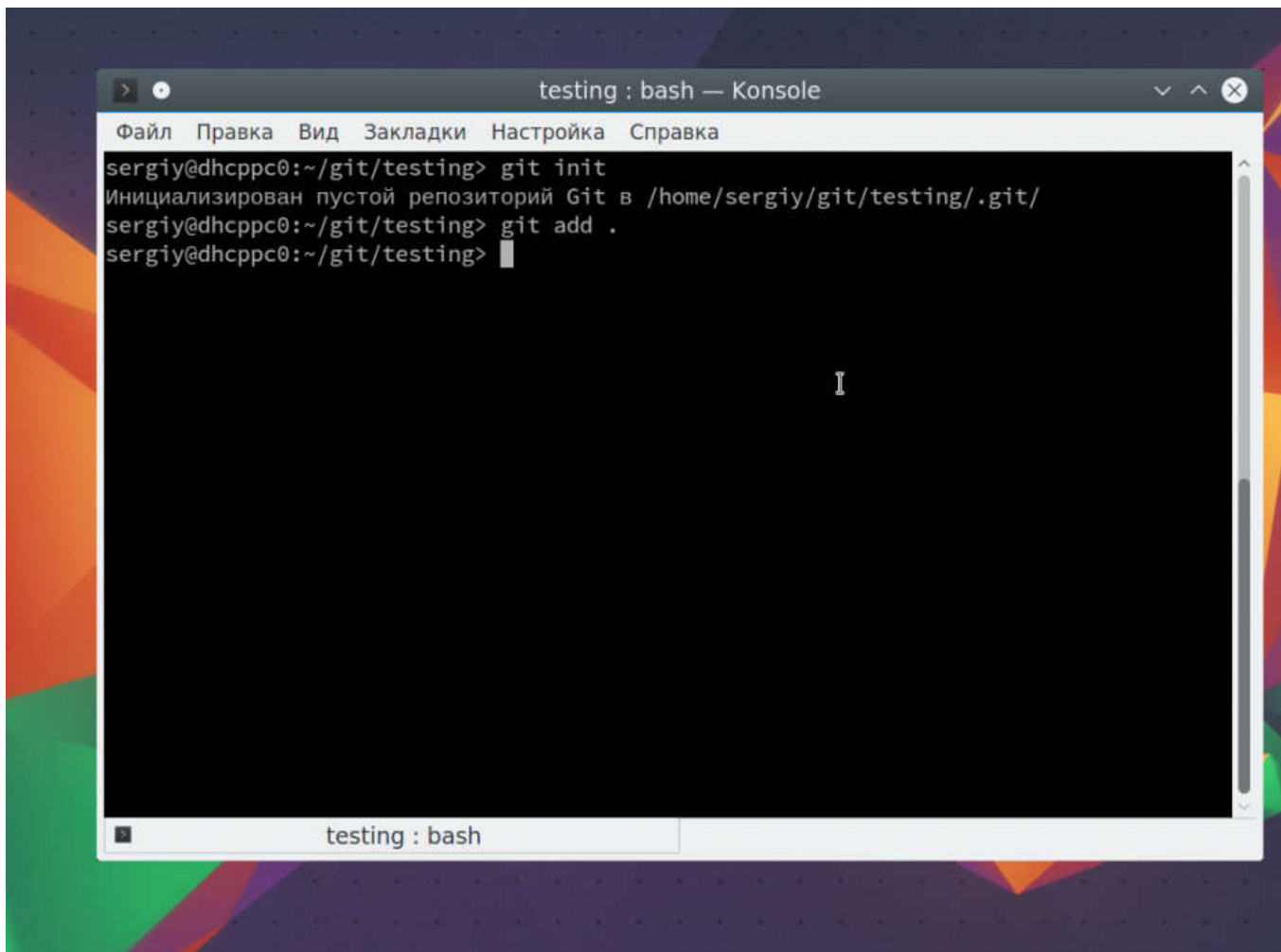
```
git init
```

A terminal window titled "testing : bash — Konsole" with a menu bar containing "Файл", "Правка", "Вид", "Закладки", "Настройка", and "Справка". The terminal shows the command "sergiy@dhcppc0:~/git/testing> git init" and its output: "Инициализирован пустой репозиторий Git в /home/sergiy/git/testing/.git/". The prompt "sergiy@dhcppc0:~/git/testing>" is followed by a cursor.

```
testing : bash — Konsole
Файл  Правка  Вид  Закладки  Настройка  Справка
sergiy@dhcppc0:~/git/testing> git init
Инициализирован пустой репозиторий Git в /home/sergiy/git/testing/.git/
sergiy@dhcppc0:~/git/testing> █
```

После того как репозиторий будет создан, вам нужно добавить свои файлы в него. Каждый файл нужно добавлять отдельно или сказать утилите, что необходимо добавить все файлы явно. Пока вы не добавите файл сам он не будет отслеживаться. Новые файлы в будущем тоже нужно добавлять, они не добавляются автоматически. Сначала добавим текущую папку:

```
git add .
```

A terminal window titled "testing : bash — Konsole" with a menu bar containing "Файл", "Правка", "Вид", "Закладки", "Настройка", and "Справка". The terminal shows the commands "sergiy@dhcppc0:~/git/testing> git init" and "sergiy@dhcppc0:~/git/testing> git add .", with the output "Инициализирован пустой репозиторий Git в /home/sergiy/git/testing/.git/" following the first command. The prompt "sergiy@dhcppc0:~/git/testing>" is followed by a cursor.

```
testing : bash — Konsole
Файл  Правка  Вид  Закладки  Настройка  Справка
sergiy@dhcppc0:~/git/testing> git init
Инициализирован пустой репозиторий Git в /home/sergiy/git/testing/.git/
sergiy@dhcppc0:~/git/testing> git add .
sergiy@dhcppc0:~/git/testing> █
```

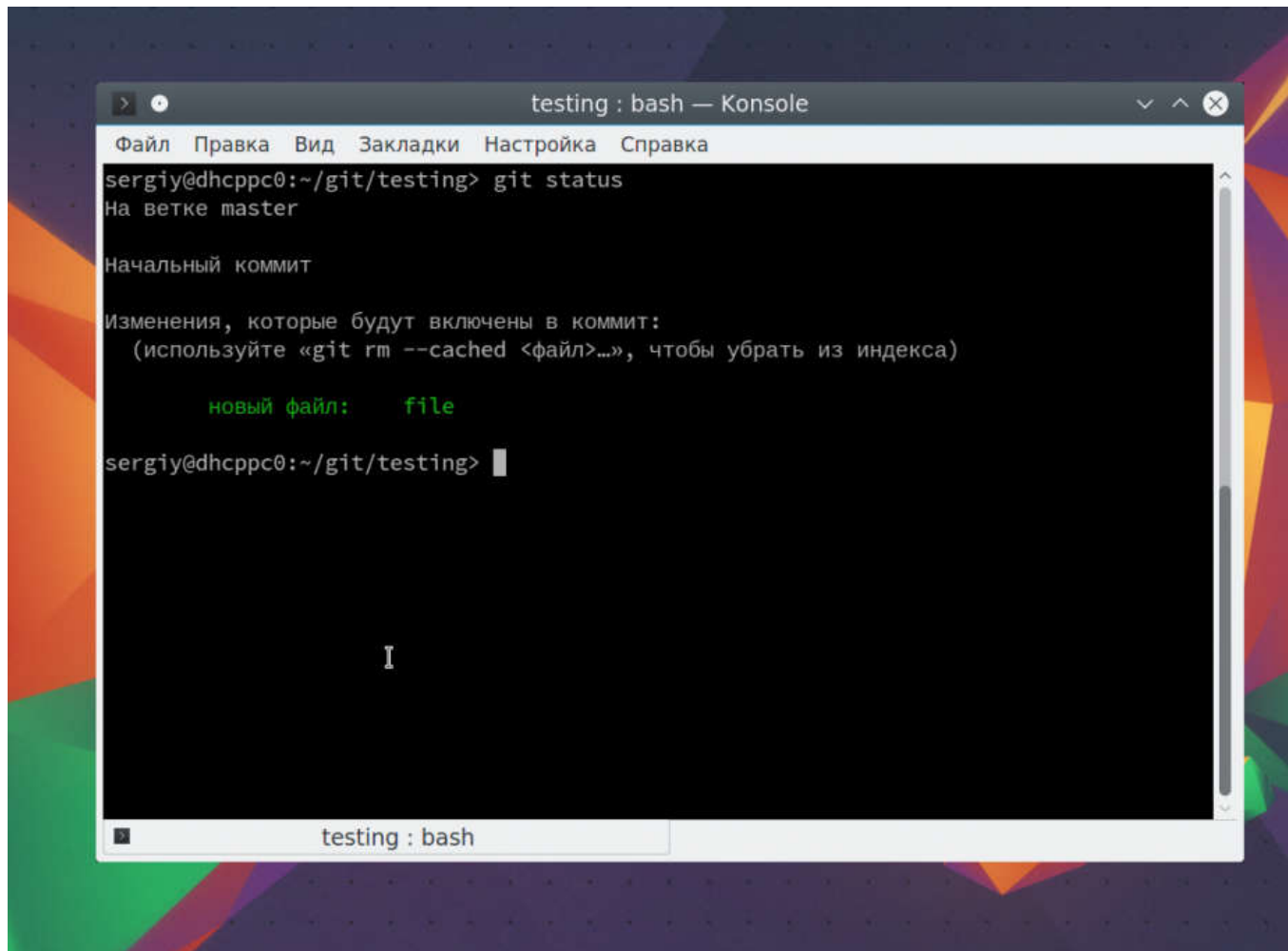
Если все прошло хорошо, то команда ничего не выведет.

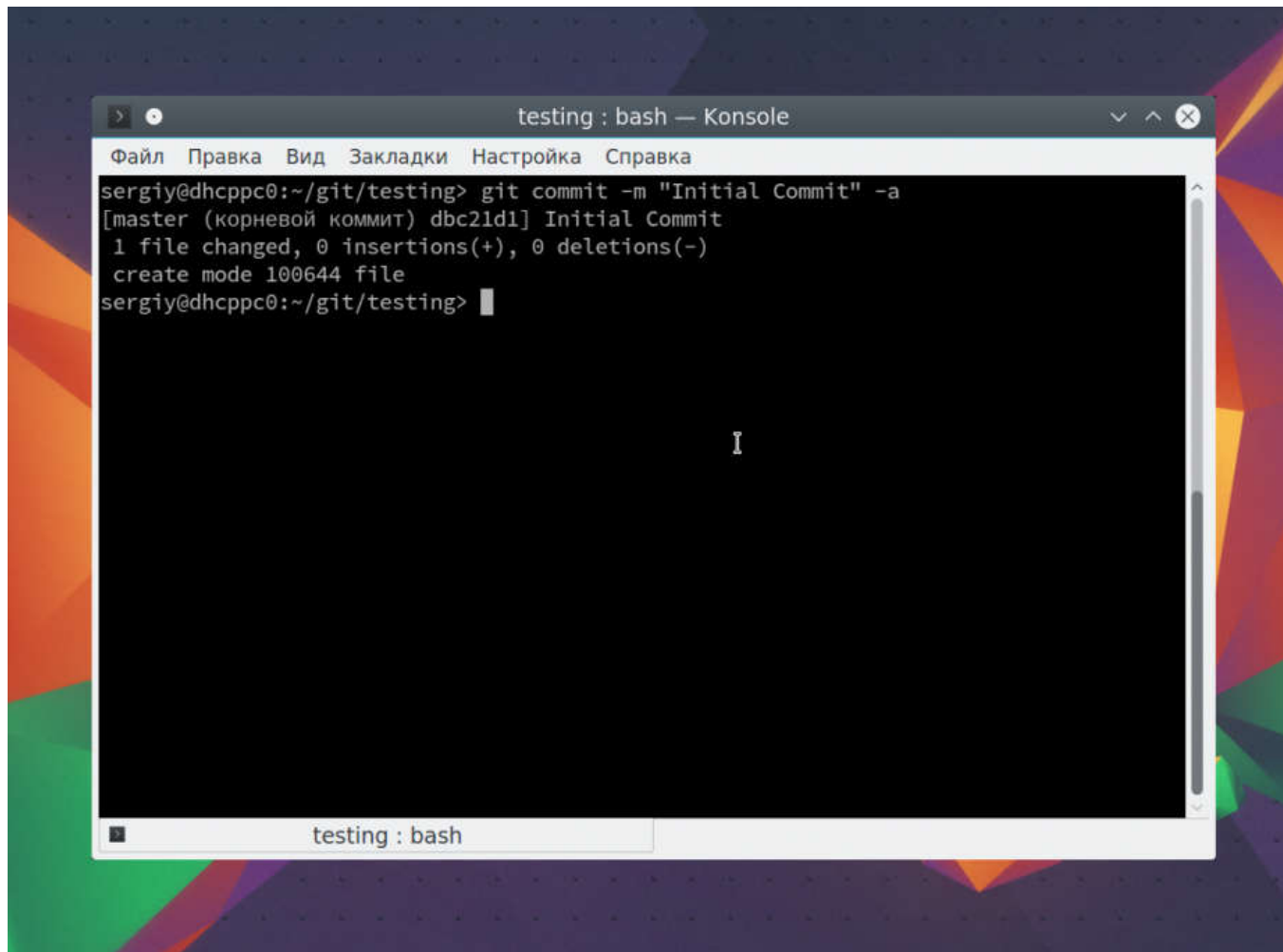
Фиксация изменений

Изменения тоже автоматически не отслеживаются. Фиксация изменений выполняется с помощью команды `commit`. Вам нужно указать что было изменено с помощью небольшого комментария, буквально в несколько предложений. Хорошая практика выполнять фиксацию перед каждым серьезным изменением.

Таким образом, вы будете хранить все версии проекта, от самой первой и до текущей, а также сможете знать что, когда и где было изменено. Чтобы создать свой первый коммит выполните:

```
git commit -m "Initial Commit" -a
```



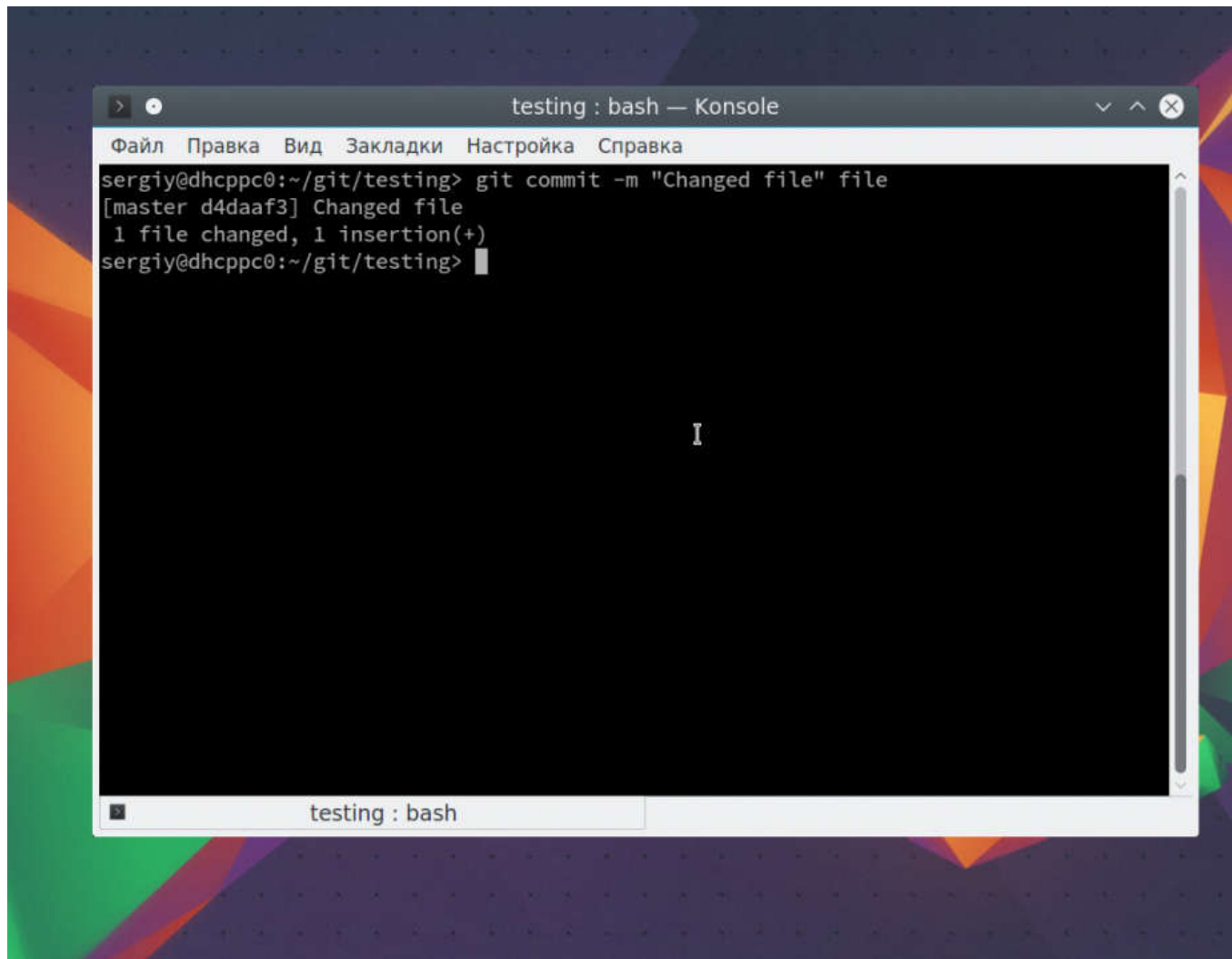
A screenshot of a terminal window titled "testing : bash — Konsole". The window has a menu bar with options: "Файл", "Правка", "Вид", "Закладки", "Настройка", and "Справка". The terminal shows the following commands and output:

```
sergiy@dhcppc0:~/git/testing> git commit -m "Initial Commit" -a
[master (корневой коммит) dbc21d1] Initial Commit
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 file
sergiy@dhcppc0:~/git/testing>
```

The terminal has a scrollbar on the right side. At the bottom of the window, there is a tab labeled "testing : bash".

Команде необходимо передать два параметра, первый — это `-m`, ваш комментарий, второй `-a`, означает, что нужно применить действие ко всем измененным файлам. Для первого раза используется этот параметр, но обычно вам нужно указать измененные файлы или каталоги. Например, можно делать так:

```
git commit -m "Changed file" file
```

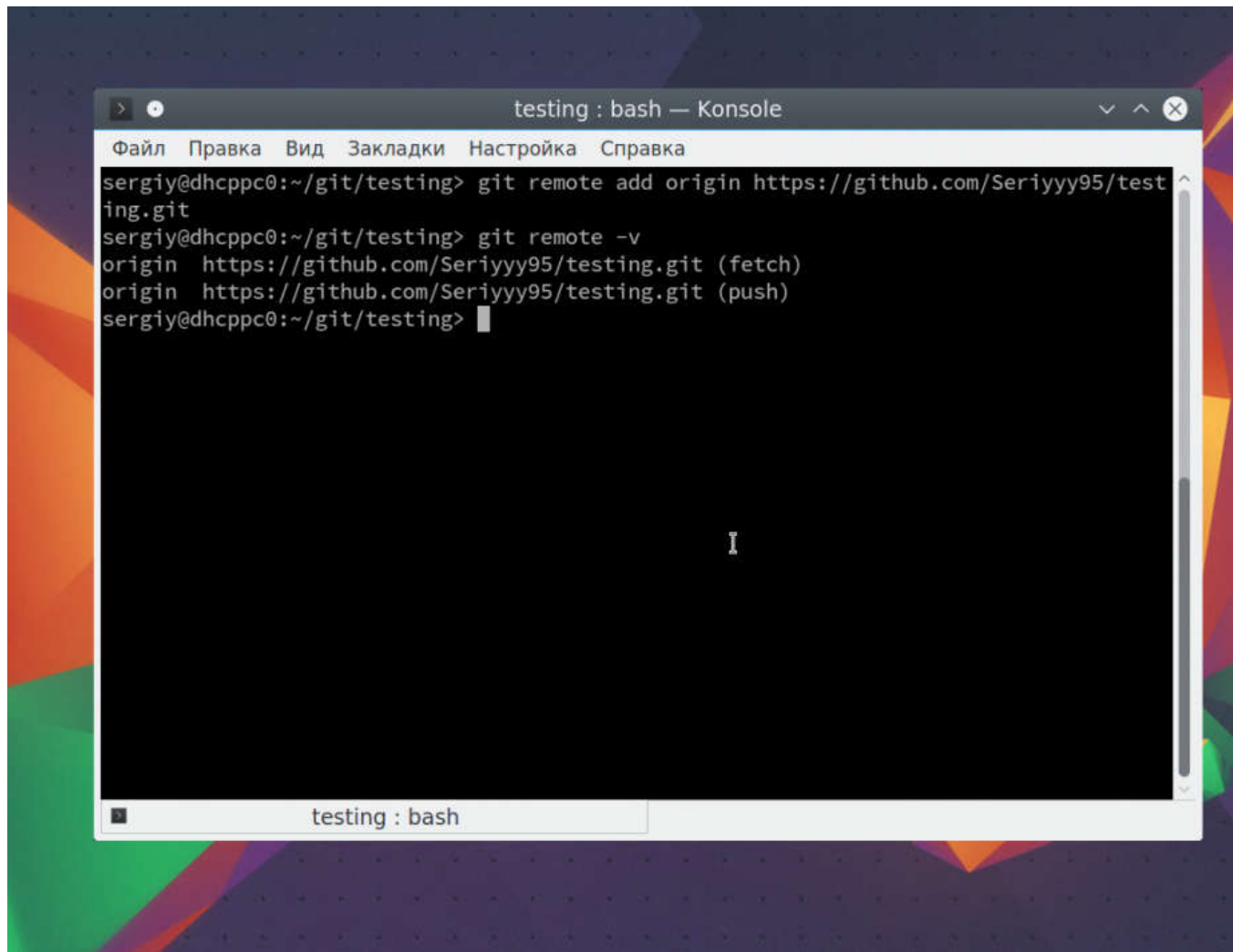



О т п р а в к а и з м е н е н и й

До этого момента мы делали все в локальном репозитории. Вы можете использовать `git` локально, если нужен только контроль версий, но иногда нужно обменяться информацией с другими разработчиками и отправить данные в удаленный репозиторий.

Сначала нужно добавить удаленный репозиторий с помощью команды `remote`. Для этого нужно передать ей URL:

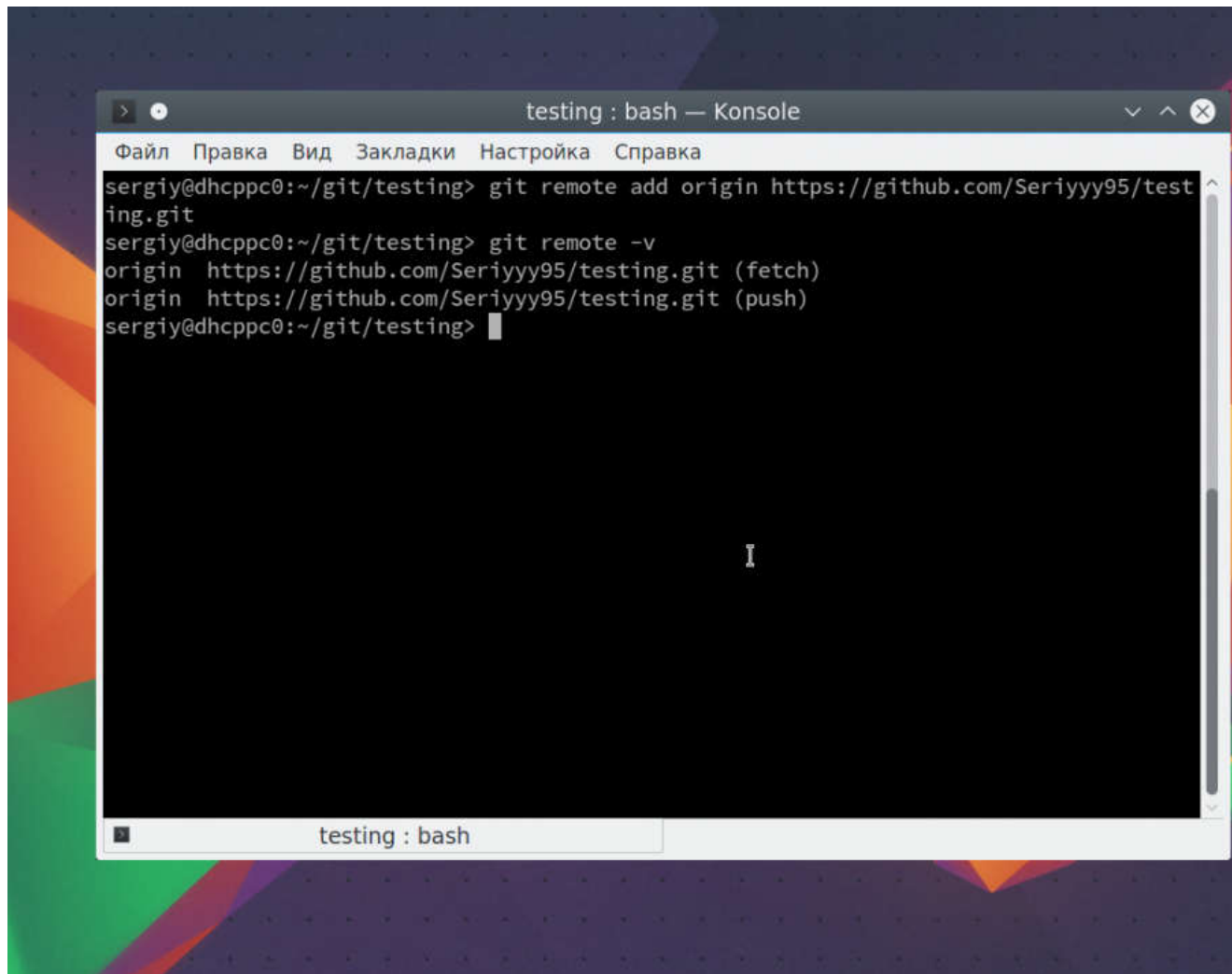
```
git remote add origin https://github.com/Seriyyy95/testing.git
```



```
testing : bash — Konsole
Файл  Правка  Вид  Закладки  Настройка  Справка
sergiy@dhcppc0:~/git/testing> git remote add origin https://github.com/Seriyyy95/testing.git
sergiy@dhcppc0:~/git/testing> git remote -v
origin  https://github.com/Seriyyy95/testing.git (fetch)
origin  https://github.com/Seriyyy95/testing.git (push)
sergiy@dhcppc0:~/git/testing>
```

Затем можно посмотреть список удаленных репозиторий:

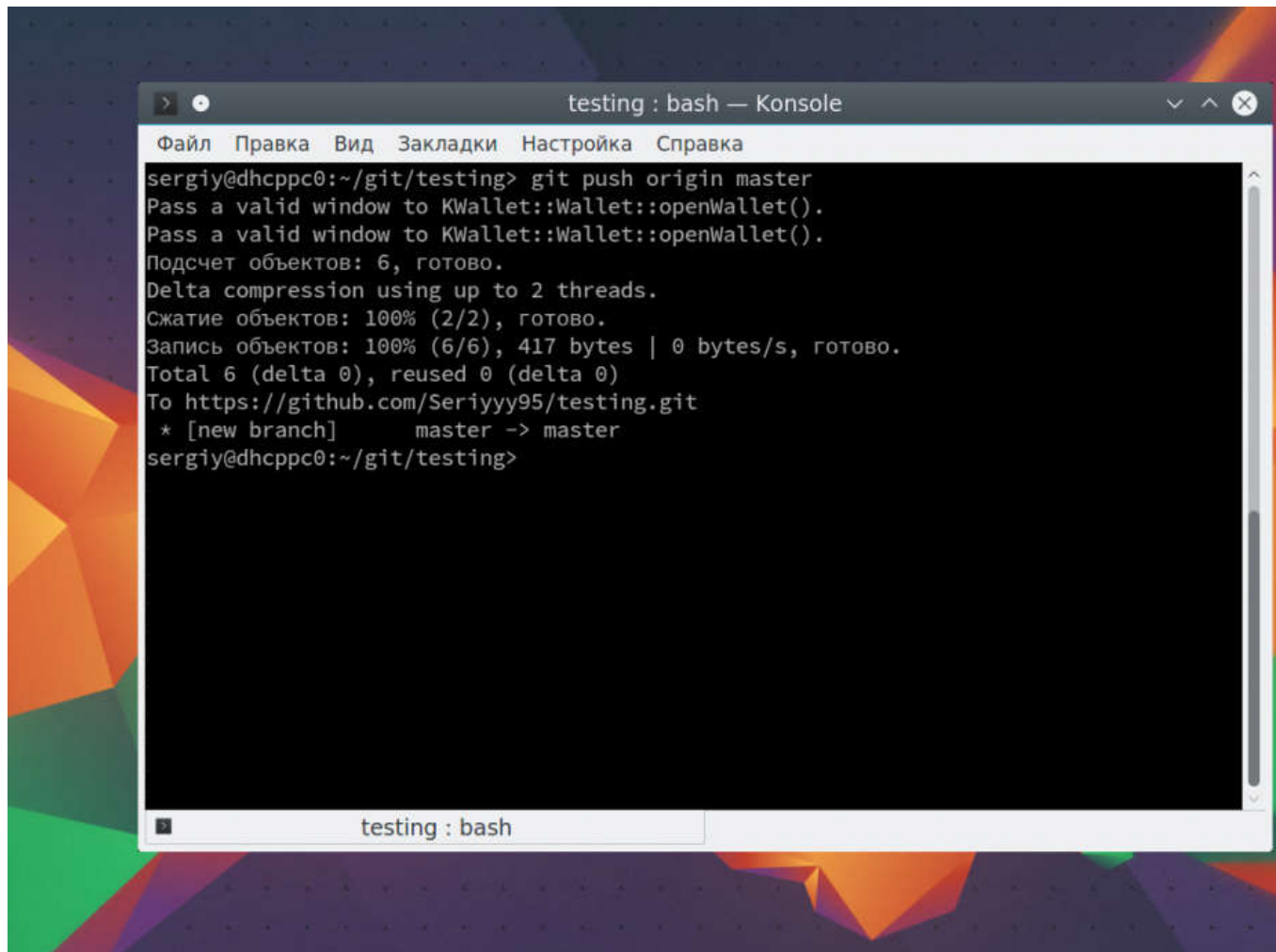
```
git remote -v
```



```
testing : bash — Konsole
Файл  Правка  Вид  Закладки  Настройка  Справка
sergiy@dhcppc0:~/git/testing> git remote add origin https://github.com/Seriyyy95/testing.git
sergiy@dhcppc0:~/git/testing> git remote -v
origin  https://github.com/Seriyyy95/testing.git (fetch)
origin  https://github.com/Seriyyy95/testing.git (push)
sergiy@dhcppc0:~/git/testing> 
```

Вы можете использовать не только github сервера, но и любые другие. Теперь для отправки ваших изменений используйте такую команду:

```
git push origin master
```



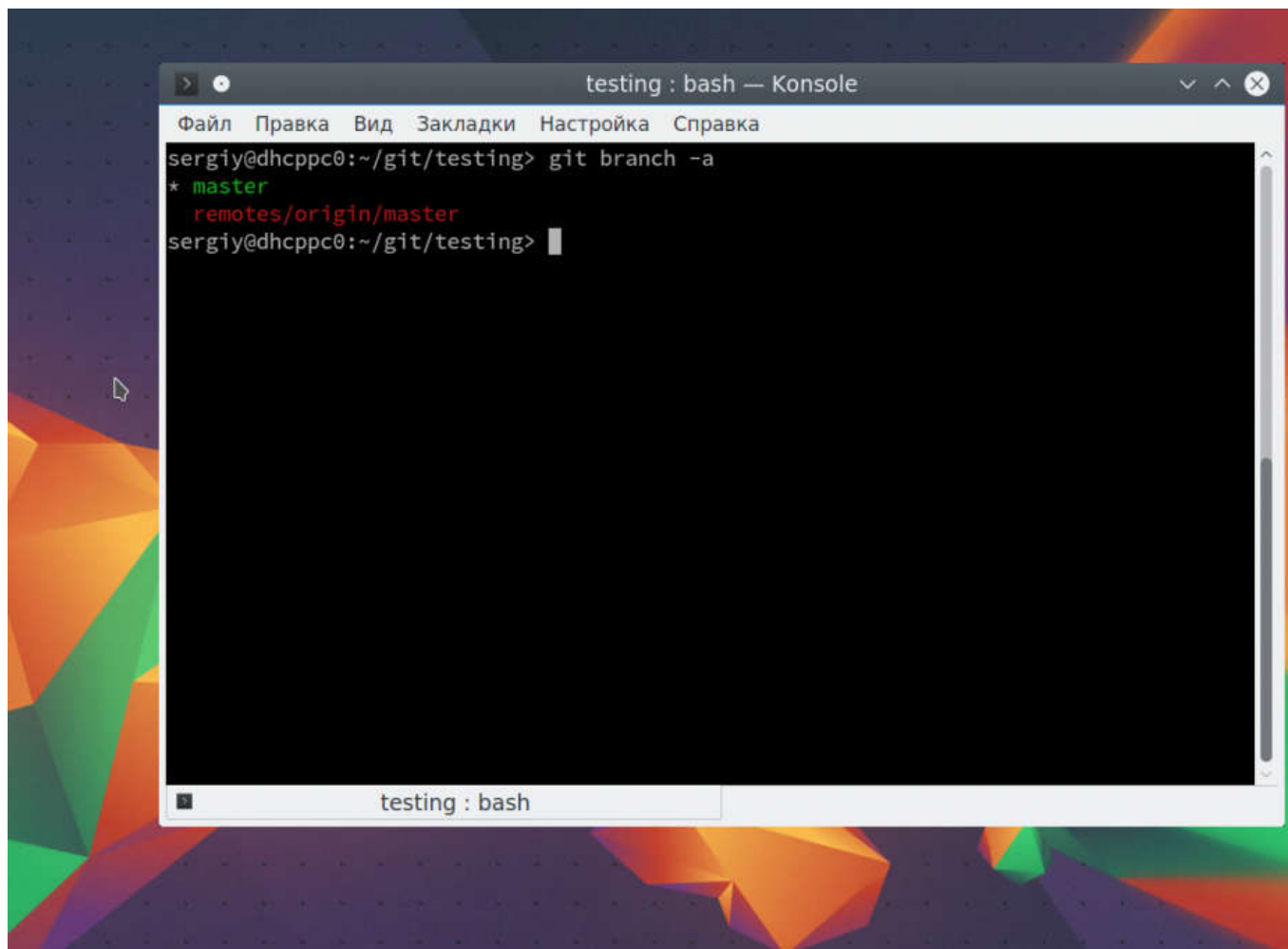
```
testing : bash — Konsole
Файл  Правка  Вид  Закладки  Настройка  Справка
sergiy@dhcppc0:~/git/testing> git push origin master
Pass a valid window to KWallet::Wallet::openWallet().
Pass a valid window to KWallet::Wallet::openWallet().
Подсчет объектов: 6, готово.
Delta compression using up to 2 threads.
Сжатие объектов: 100% (2/2), готово.
Запись объектов: 100% (6/6), 417 bytes | 0 bytes/s, готово.
Total 6 (delta 0), reused 0 (delta 0)
To https://github.com/Seriy95/testing.git
 * [new branch]      master -> master
sergiy@dhcppc0:~/git/testing>
```

Команда push указывает, что нужно отправить данные в удаленный репозиторий, origin — наш настроенный репозиторий, а master — ветвь.

У п р а в л е н и е в е т в я м и

Для простых проектов достаточно одной ветви. Но если проект большой и он имеет несколько версий, в том числе тестовую, то может понадобиться создать для каждой из них отдельную ветвь. Сначала смотрим доступные ветви:

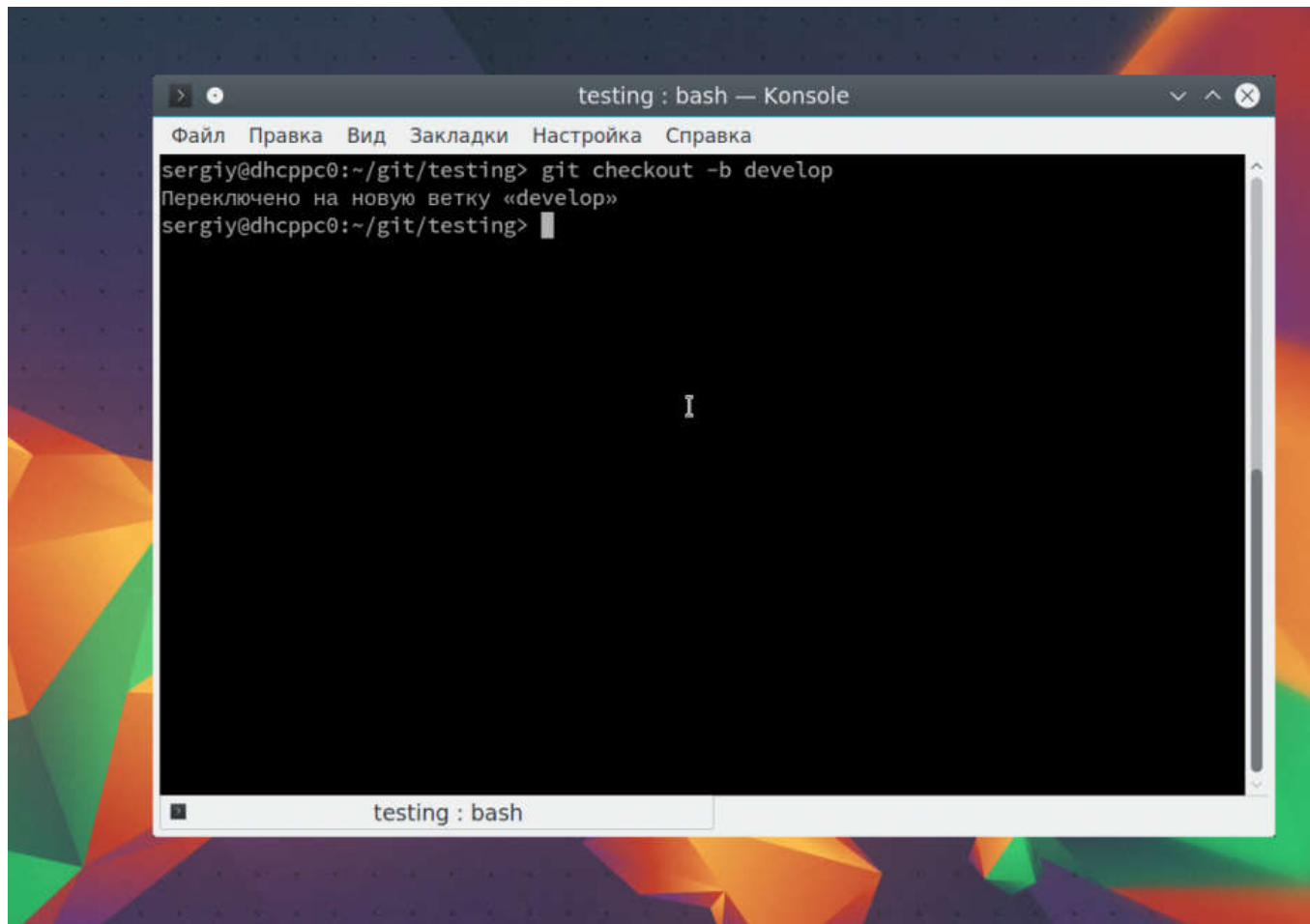
```
git branch -a
```

A screenshot of a terminal window titled "testing : bash — Konsole". The window has a menu bar with "Файл", "Правка", "Вид", "Закладки", "Настройка", and "Справка". The terminal shows the command "sergiy@dhcppc0:~/git/testing> git branch -a" and its output: "* master" and "remotes/origin/master". The prompt "sergiy@dhcppc0:~/git/testing>" is followed by a cursor. The terminal window is set against a dark background with a colorful, abstract geometric pattern on the left side.

```
testing : bash — Konsole
Файл Правка Вид Закладки Настройка Справка
sergiy@dhcppc0:~/git/testing> git branch -a
* master
  remotes/origin/master
sergiy@dhcppc0:~/git/testing> 
```

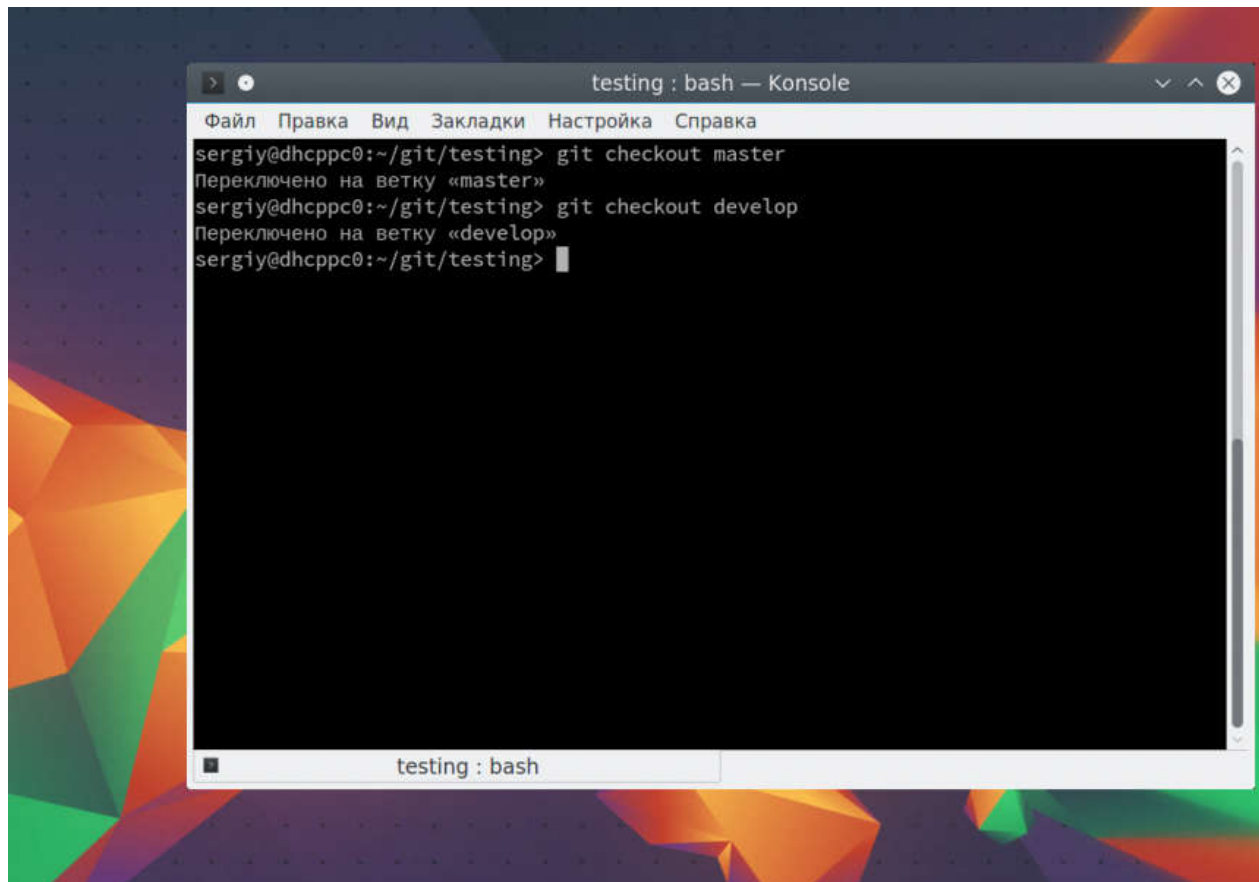
Опция -a указывает что нужно вывести все ветви, даже не синхронизированные. Звездочка указывает на активную ветвь. Теперь создадим ветвь для разработки с помощью команды checkout:

```
git checkout -b develop
```



Переключаться между ветвями можно тоже с помощью той же команды:

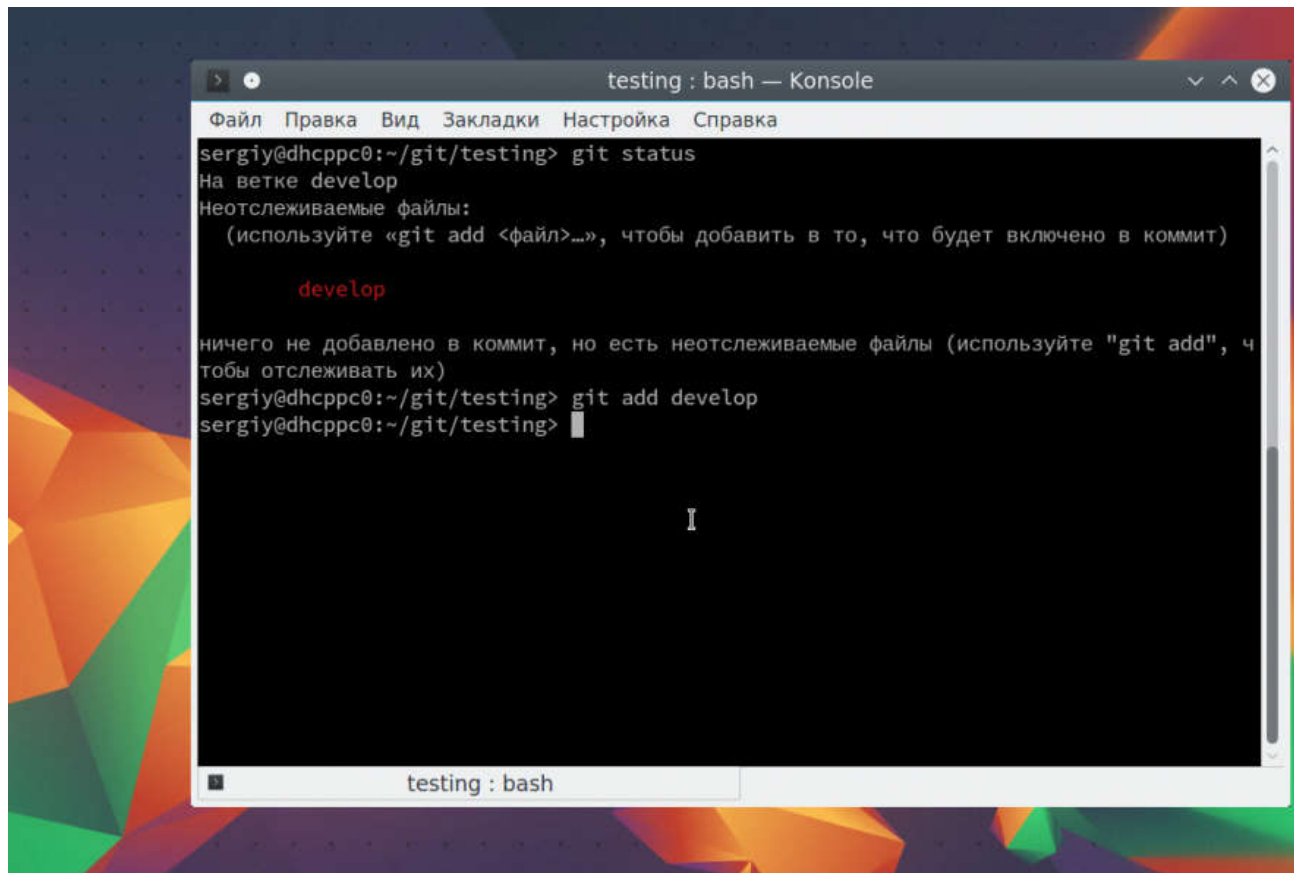
```
git checkout master  
$ git checkout develop
```



```
testing : bash — Konsole
Файл  Правка  Вид  Закладки  Настройка  Справка
sergiy@dhcppc0:~/git/testing> git checkout master
Переключено на ветку «master»
sergiy@dhcppc0:~/git/testing> git checkout develop
Переключено на ветку «develop»
sergiy@dhcppc0:~/git/testing> 
```

Теперь создадим еще один файл:

```
touch develop
```



```
testing : bash — Konsole
Файл  Правка  Вид  Закладки  Настройка  Справка
sergiy@dhcppc0:~/git/testing> git status
На ветке develop
Неотслеживаемые файлы:
  (используйте «git add <файл>...», чтобы добавить в то, что будет включено в коммит)

    develop

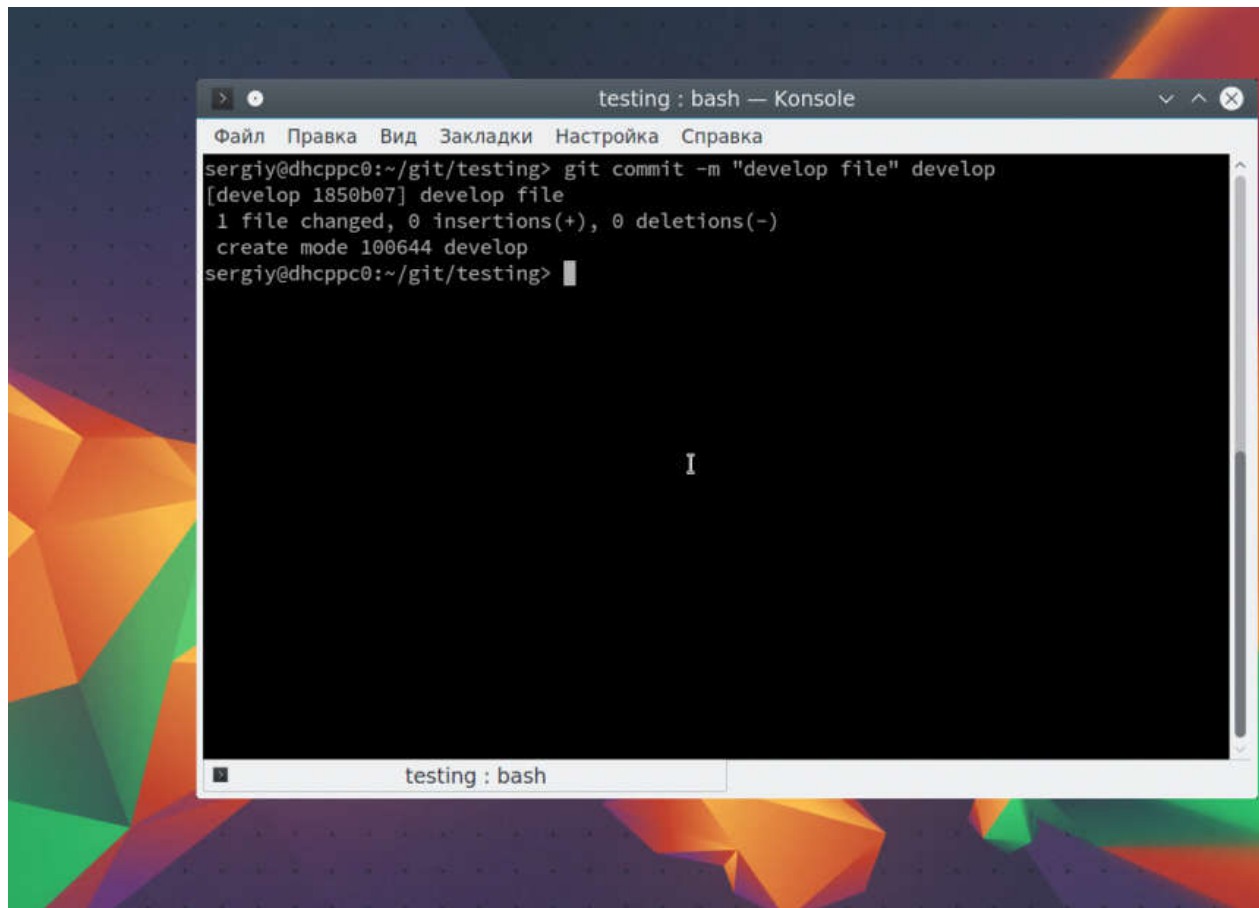
ничего не добавлено в коммит, но есть неотслеживаемые файлы (используйте "git add", чтобы отслеживать их)
sergiy@dhcppc0:~/git/testing> git add develop
sergiy@dhcppc0:~/git/testing> 
```

И добавим его в нашу новую ветвь develop:

```
git add develop
```

Сделаем коммит для внесенных изменений:

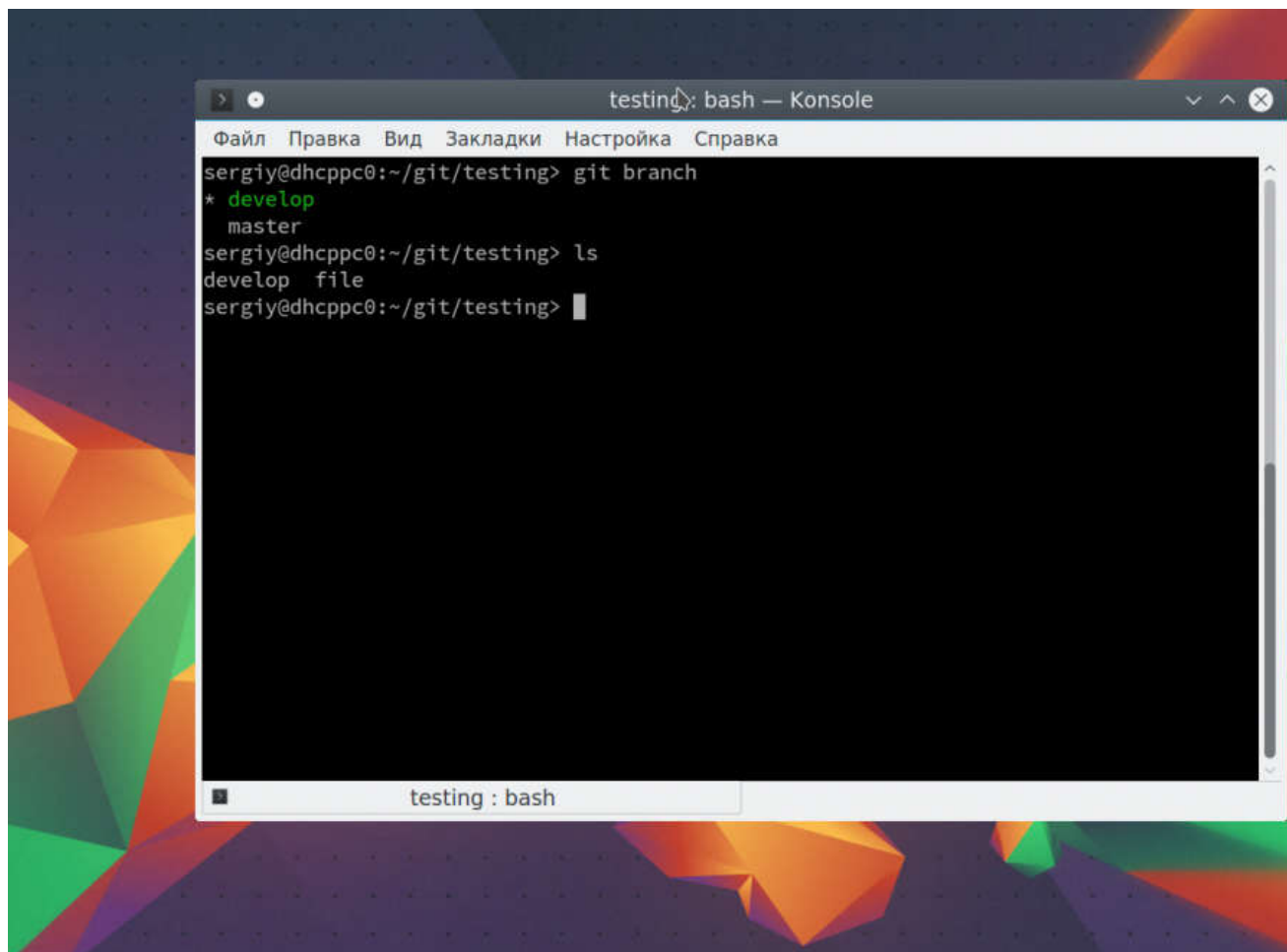
```
git commit -m "develop file" develop
```

```
testing : bash — Konsole
Файл  Правка  Вид  Закладки  Настройка  Справка
sergiy@dhcppc0:~/git/testing> git commit -m "develop file" develop
[develop 1850b07] develop file
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 develop
sergiy@dhcppc0:~/git/testing> 
```

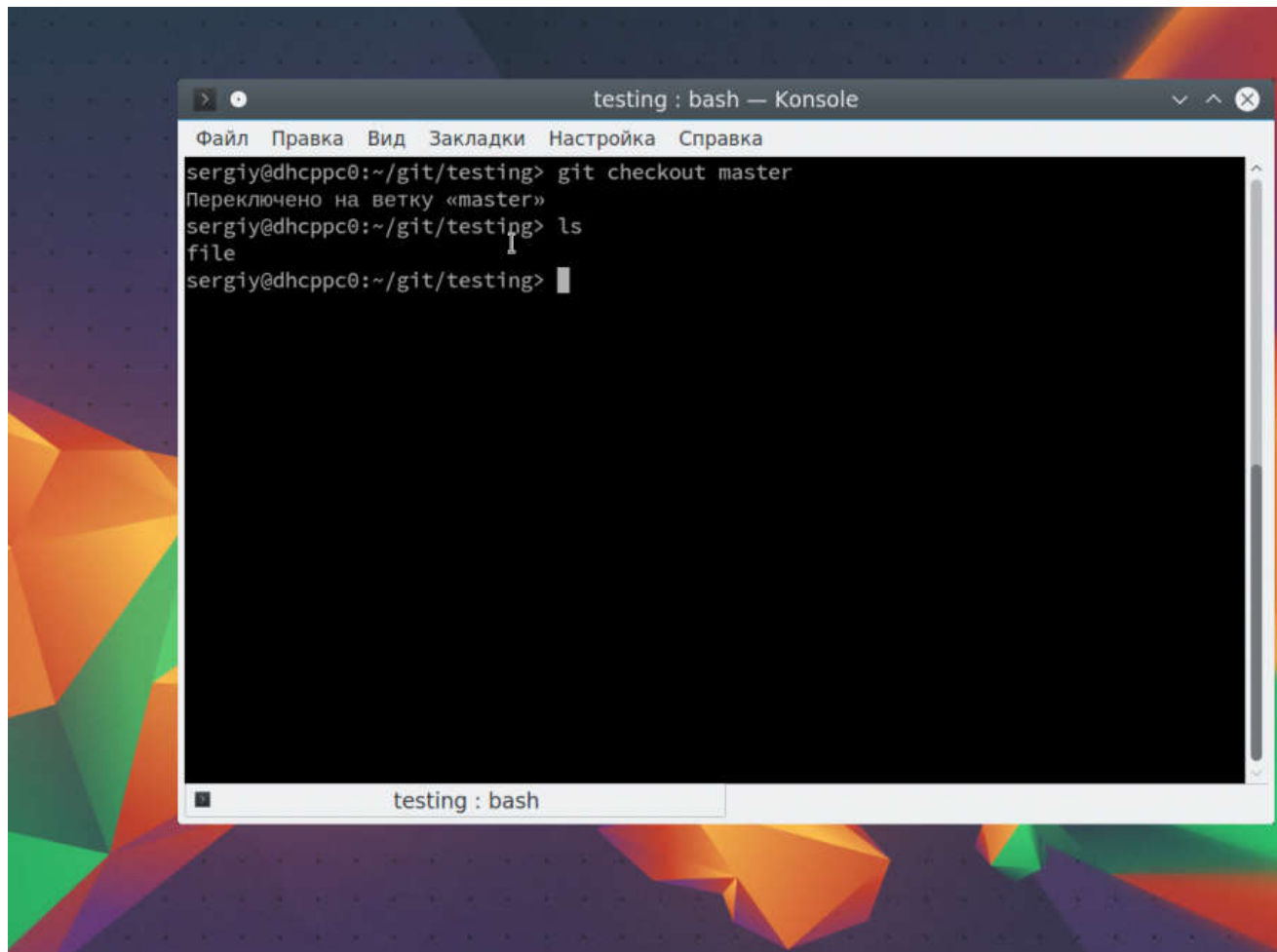
Дальше проверим существует ли этот файл в основной ветке master или только в дополнительной. Смотрим текущую ветку:

```
git branch
$ ls
```



Затем переключаемся на ветку master и снова смотрим:

```
git checkout master  
$ git branch  
$ ls
```

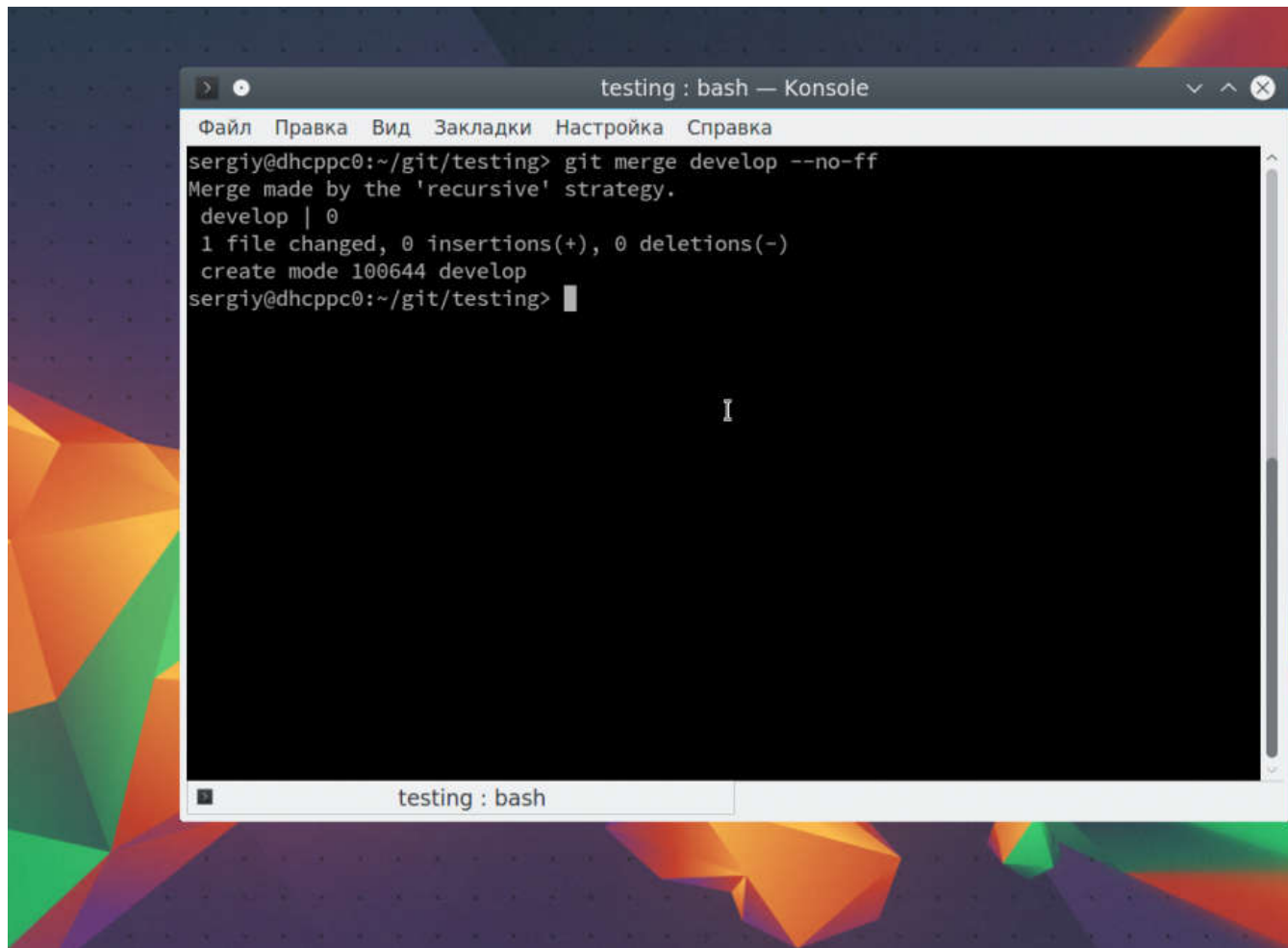
A screenshot of a terminal window titled "testing : bash — Konsole". The window has a menu bar with "Файл", "Правка", "Вид", "Закладки", "Настройка", and "Справка". The terminal shows the following commands and output:

```
sergiy@dhcppc0:~/git/testing> git checkout master
Переключено на ветку «master»
sergiy@dhcppc0:~/git/testing> ls
file
sergiy@dhcppc0:~/git/testing> 
```

The terminal background is black, and the text is white. The window is set against a colorful, abstract geometric background.

Здесь файла нет, так и должно быть. В git есть такая полезная вещь, как слияние. С помощью нее вы можете объединить две ветви. Например, переместить код из рабочей ветки в стабильную. Для этого достаточно выполнить команду merge:

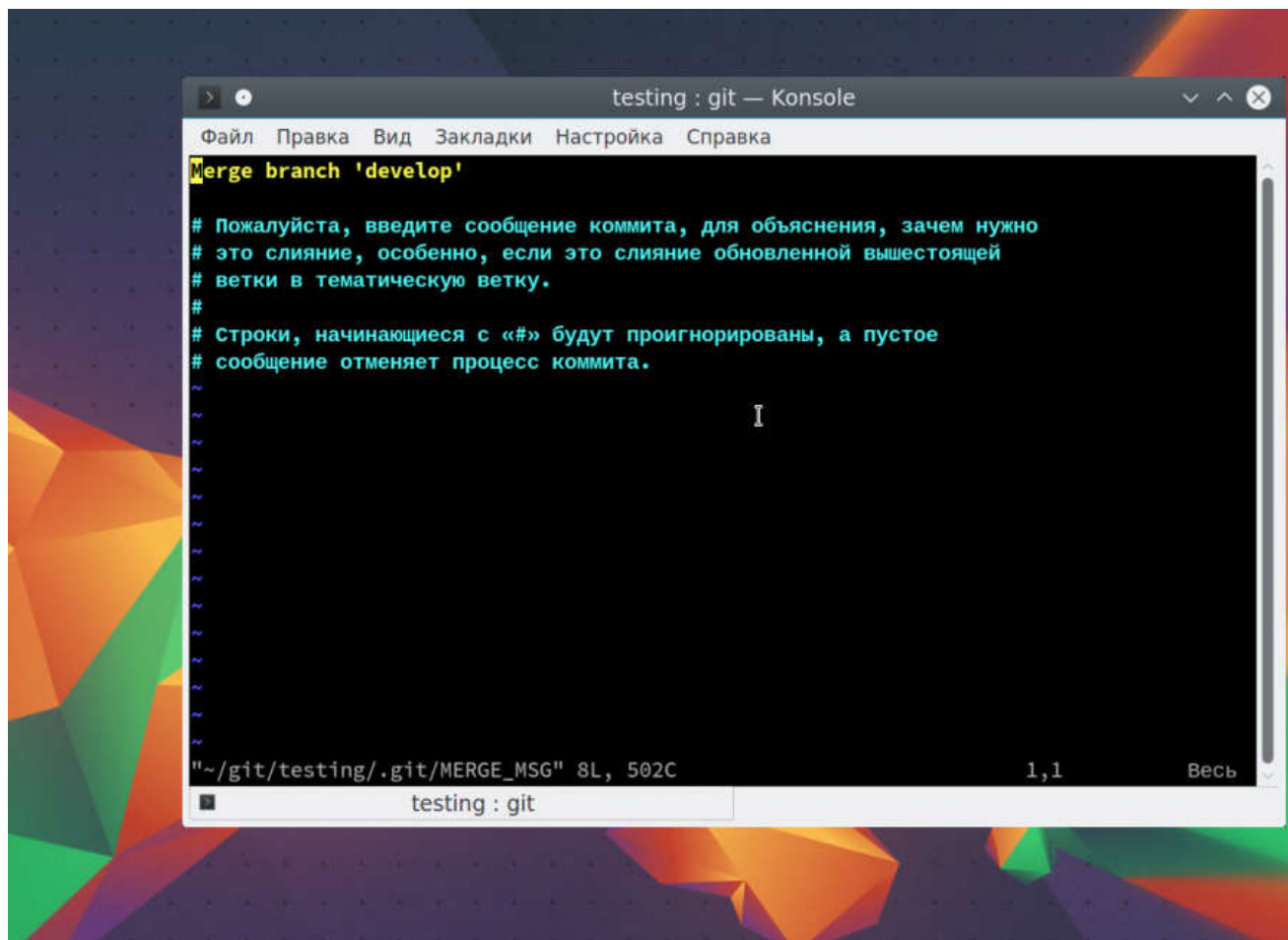
```
git merge develop --no-ff
```

A screenshot of a terminal window titled "testing : bash — Konsole". The window has a menu bar with "Файл", "Правка", "Вид", "Закладки", "Настройка", and "Справка". The terminal shows the following text:

```
sergiy@dhcppc0:~/git/testing> git merge develop --no-ff
Merge made by the 'recursive' strategy.
 develop | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 develop
sergiy@dhcppc0:~/git/testing>
```

The terminal has a dark background with a light-colored cursor. The window's title bar and menu bar are light gray. The background of the desktop is a colorful, abstract geometric pattern.

Перед тем как будет выполнено слияние вам нужно ввести комментарий, зачем это нужно. Затем если вы еще раз выполните ls, то увидите, что здесь уже есть нужный файл. Наши примеры git подошли к концу.



В ы в о д ы

В этой статье мы рассмотрели как пользоваться git для управления версиями своих проектов. Это только самая основная информация, и система контроля версий git может еще очень многое, но рассмотрение его дополнительных возможностей выходит за рамки данной статьи. Надеюсь, эта статья была вам полезной.

П о х о ж и е з а п и с и :

Нет похожих записей

О ц е н и т е с т а т ь ю :

★★★★★ (11 оценок, среднее: 4,91 из 5)

[1 Сохранить](#)

О б а в т о р е



[admin](#)

Основатель и администратор сайта losst.ru, увлекаюсь открытым программным обеспечением и операционной системой Linux. В качестве основной ОС сейчас использую Ubuntu. Кроме Linux интересуюсь всем, что связано с информационными технологиями и современной наукой.

В а ш е и м я т о ж е м о ж е т б ы т ь з д е с ь . Н а ч н и т е
п и с а т ь с т а т ь и д л я L o s s t . Э т о п р о с т о !
С м о т р и т е п о д р о б н е е к а к н а ч а т ь п и с а т ь
с т а т ь и – [П и ш и т е д л я н а с](#)

7 к о м м е н т а р и е в

1.



[Sergei Shekin](#) Я н в а р ь 24, 2017

[О т в е т и т ь](#)

2.

Спасибо за статью, подумываю перенести кое что в ГИТ, но точно в терминале работать не буду, как на счет обзора программы GitEye или GitKraken?

Это было бы бомбой для меня, очень интересно.

3.



[Misha Goodov](#) Ф е в р а л ь 10, 2017

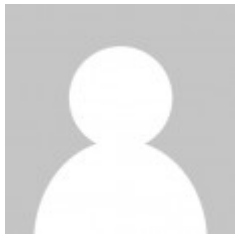
[О т в е т и т ь](#)

○

Тоже GitKraken использую. По сути всё есть на оф.сайте, но было бы круто почитать переводы))

○

4.



[Sergei Shekin](#) Я н в а р ь 24, 2017

[О т в е т и т ь](#)

5.

Думаю актуальнее GitEye, она и мощная и бесплатная

6.

7.



Camino Я н в а р ь 25, 2017 [О т в е т и т ь](#)

8.

git gui тоже неплох

9.

10.



[Igor Stetsyuk](#) Я н в а р ь 25, 2017

[О т в е т и т ь](#)

11.

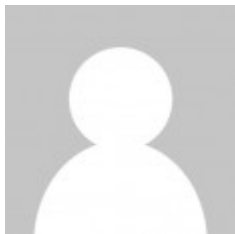
первая статья которая понравилась, спасибо.
от себя добавлю:
для удобства работы в консоле есть вот такая штука git-aware-prompt
еще очень удобно в башре добавлять чет на подобии

12.

```
example () {  
cd ~/gitlab/example  
}
```

13.

14.



Н и к о л а й М а й 10, 2018 [О т в е т и т ь](#)

15.

Спасибо тебе, Ты крут, без подхалимажа. Твоя мысль четкая и структуриализированная.
Тема не такая уж заоблачной сложности, но подавляющее большинство несут полный сумбур, концами своей мысли утопая в болоте.
Легкого тебе дня!

16.

17.



К и р и л л М а й 22, 2018 [О т в е т и т ь](#)

18.

worktree — управление !!деревнями! разработки.

19.

О т в е т и т ь