Финальный тест по GIT.  
  
1) Соедините тип системы контроля версий с верным определением: (1 балл)  
  
1. В локальных системах контроля версий   
2. В централизованных системах контроля версий  
3. В распределенных системах контроля версий

a. Разработчики подключаются к общему серверу и отправляют туда свои изменения.  
b. У разработчиков есть копия основного репозитория и могут мержить и добавлять туда свои изменения.  
c. Все изменения разработчиков хранятся только на их локальных машинах.

**1 – c**

**2 – a**

**3 – b**

2) Опишите, что такое Git, Github, Gitlab. (1 балл)

**Git** – система контроля версий, которая отслеживает изменения в файлах.

**Github и Gitlab** – платформы для управления проектами и репозиториями программного кода. Они помогают командам совместно разрабатывать программное обеспечение. Одно из главных отличий этих платформ друг от друга в том, что в Gitlab есть комплексный инструмент DevOps.

3) Соедините Git статус с его описанием: (1 балл)

1. Modified  
2. Commited  
3. Changed  
4. Staged  
a. Состояние файла уже существует в гите и он не изменялся.  
b. Файл был изменен, но изменение не было добавлено в индекс.  
c. Файл был изменен и добавлен в индекс.  
d. Такой статус в гите не существует.

**1 – b**

**2 – a**

**3 – d**

**4 – c**

4) Ситуация: Вы и ваши коллеги используете централизованный репозиторий, куда отправляете свои изменения. Вы решили сделать большинство изменений в локальном репозитории, а ветка “main” в удаленном репозитории должна будет содержать финальные изменения. Опишите, как вы будете отправлять ваши изменения в ветку “main” в удаленный (remote) репозиторий. (1 балл)

1 – с помощью команды git clone склонирую удаленный репозиторий в свой локальный

2 – создам свою ветку (git branch my\_branch)

3 – после необходимых изменений отправлю свою ветку на удаленный репозиторий (git push --set-upstream origin my\_branch)

4 – после утверждения коллегами сделаю merge слияние изменений из своей ветки в ветку main

5) Опишите ситуацию когда могут возникнуть конфликты слияния (merge conflicts). (1 балл)

Два разработчика изменили одну и ту же строку одного и того же файла на разных ветках. Git не сможет определить какое изменение оставить, потребуется вмешательство разработчика.

6) Опишите пути решения конфликтов слияния. (1 балл)

1 – открыть с помощью текстового редактора нужный файл, в котором Git пометил изменения, и вручную можно удалить не нужные изменения.

2 – с помощью mergetool выбрать одно конкретное изменение, оставить оба или написать третий вариант

3 – отменить слияние с помощью команды git merge --abort

4 – выбрать одну версию изменений с помощью команд git checkout --ours (выбор текущей ветки) git checkout --theirs (выбор из объединяемой ветки)

7) Опишите что такое SSH ключ и для чего он нужен. Как мы можем его сгенерировать? (1 балл)

**SSH ключ** – это пара криптографический ключей, используемых для безопасного подключения к удаленным серверам и для аутентификации пользователей. Он состоит из двух частей: приватный (не должен никому передаваться), публичный (может быть передан)

Для генерации нужно открыть терминал и ввести команду ssh-keygen -t rsa -b 4096 -C "my\_email"

8) Соедините Unix команды с их действиями: (1 балл)

1.cat 2.cp  
3.ls 4.mkdir  
5.rm 6.cd  
7.grep 8.mv  
a. Просмотр содержимого файла в консоли.

b. Переместить файл  
c. Перейти в директорию по пути

d. Поиск по паттерну или строке   
e. Создать новую директорию

f. Удалить файл

g. Посмотреть список объектов в директории

h. Копировать файл

**1 – a**

**2 – h**

**3 – g**

**4 – e**

**5 – f**

**6 – c**

**7 – d**

**8 – b**

9) Как вы вызовете подсказку в гите по конкретной команде? (1 балл)

git help command

git command --help

10) Что такое Vim? Как выйти из Vim? (1 балл)

Vim - это текстовый редактор. Нужно нажать Esc затем :wq – сохранить и выйти или :q! выйти без сохранения. Если изменения не были внесены, то выйти можно через :q

11) Ситуация:Различные скрипты создают в вашей директории файлы логов. Вы хотите настроить гит на игнорирование всех “\*.log” файлов в вашей директории и всех поддиректориях, но не хотите игнорировать файл “monthly\_test.log”, который важен для вас. Опишите, что нужно добавить в “.gitignore”. (1 балл)

#Игнорировать все файлы .log

\*\*.log

#Исключить их игнорирование файл monthly\_test.log

!monthly\_test.log

12) Ситуация:Вы вчера закоммитили файл “huge.txt”. Сегодня вы видите, что он был изменен и изменения добавлены в индекс. Напишите команды, которые вы будете использовать для:

a) Просмотра изменений, которые были внесены в файл “huge.txt”

**git diff huge.txt**

b) Для удаления всех изменений в файле из индекса. (1 балл)

**git restore –staged huge.txt**

13) Опишите назначение команды git rebase -i HEAD~3 и напишите какие операции могут быть осуществлены с помощью этой команды над историей коммитов. (1 балл)

git rebase -i HEAD~3 позволяет изменить 3 последних коммита. При выполнении данной операции открывается интерфейс редактирования с помощью которого можно

pick — Оставить коммит без изменений.

reword — Изменить сообщение коммита без изменения его содержимого.

edit — Внести изменения в сам коммит (например, добавить, изменить, удалить файлы или исправить ошибки в коде).

squash — Объединить (слить) этот коммит с предыдущим. Сообщения коммитов будут объединены.

fixup — Аналогично squash, но сообщения коммитов не объединяются; сообщение будет взято только из предыдущего коммита.

drop — Удалить коммит из истории.

exec — Выполнить произвольную команду shell после завершения операции для коммита.

14) Опишите, что такое “fork” и как вы можете отправить изменения в него. (1 балл)

Fork — это копия чужого репозитория, созданная в вашем аккаунте. Fork позволяет работать с проектом, не влияя напрямую на основной репозиторий.

1 – создание форка: нужно перейти на репозиторий и нажать кнопку форк

2 – клонирование форка в локальный репозиторий: git clone url\_fork

3 – отправка изменений в форк: git push origin my\_branch

15) Опишите разницу между легковесным и аннотированным тегом. Приведите пример команды для создания каждого из них. (1 балл)

annotated хранит дополнительную информацию, например комментарий, а light-weight – нет

annotated: git tag -a <tagname> -m "<message>"

light\_weight: git tag <tagname>

16) Как вы создаете новую ветку в своем локальном репозитории? Как вы отправляете ее в удаленный репозиторий? (1 балл)

git branch my\_branch или git checkout -b my\_branch (позволяет создать и сразу перейти на ветку)

git push -set--upstream origin my\_branch

17) Что такое “cherry-pick” в гите? (1 балл)

Команда, которая позволяет перенести конкретный коммит в текущую ветку из другой ветки

git cherry-pick <commit-hash>

18) Опишите, что делает команда git stash. Какая разница между git stash apply и git stash pop? (1 балл)

Команда git stash временно сохраняет текущие изменения рабочей директории и индекса, убирая их из рабочего дерева и откатывая ветку к последнему зафиксированному состоянию. Это удобно, если нужно переключиться на другую ветку или выполнить другие действия, не добавляя текущие изменения в коммит.

git stash apply: применяет последние сохраненные изменения из stash к текущей рабочей директории, но оставляет копию этих изменений в stash.

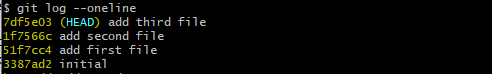
git stash pop: применяет последние (или указанные) сохраненные изменения из stach к рабочей директории и одновременно удаляет их из stash.

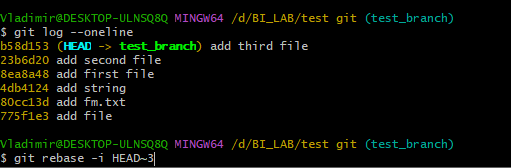
19) Какая разница между git pull и git fetch? (1 балл)

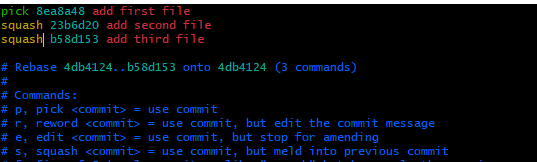
git fetch - загружает изменения из удаленного репозитория, не объединяя их с локальной веткой

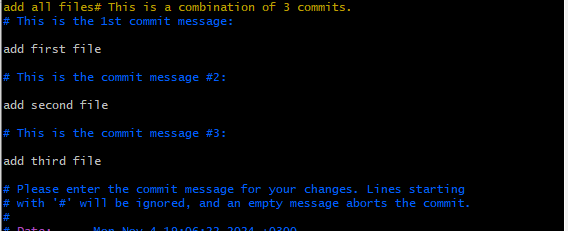
git pull - загружает изменения и сразу объединяет их с текущей веткой

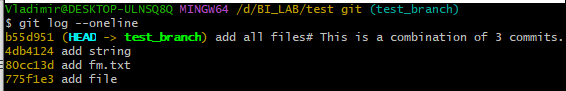
20) Как вы объединяете/комбинируете несколько коммитов в один? (2 балла)  
Приведите пример объединения 3 последних коммитов в один:











21) Когда вы клонируете репозиторий какое имя по-умолчанию у вашего удаленного репозитория? Как можно добавить другой удаленный репозиторий? (1 балл)

При клонировании репозитория Git по умолчанию присваивает уделенному репозиторию имя **origin**. Это имя ссылается URL удаленного репозитория.

С помощью команды git remote add <name> <url> можно добавить дополнительный удаленный репозиторий

22) Существует ли способ откатить коммит, который был уже отправлен в удаленный репозиторий? Если это возможно – опишите как. (2 балл)

Откатить коммит, который уже был отправлен в удаленный репозиторий можно с помощью команд

1 - git revert <commit hash>

2 - git reset --hard <commit hash>

Первый способ предпочтительнее и безопаснее, так как он не изменяет историю, а просто создает новый коммит, который "отменяет" изменения, сделанные в отправленном коммите, в отличие от второго, который изменяет историю.

23) Какая разница между ‘HEAD’, ‘working tree’ и ‘index’? (1 балл)

**HEAD** — это ссылка на текущую ветку или коммит, на котором вы находитесь в данный момент. Обычно он указывает на последний коммит в текущей ветке.

**Working tree** — это директория, в которой находятся файлы вашего проекта, которые вы можете редактировать. Это текущее состояние ваших файлов в файловой системе.

**Index** — это промежуточное хранилище, в котором вы указываете, какие изменения вы хотите включить в следующий коммит.

24) Что делает команда git revert? (1 балл)

Команда git revert используется для создания нового коммита, который отменяет изменения, внесенные ранее в указанном коммите.

25) Опишите способ, как можно добавить изменения в последний коммит. (1 балл)

Чтобы добавить изменения в последний коммит в Git, можно использовать команду git commit --amend. Эта команда позволяет изменить последний коммит, добавив к нему новые изменения или изменив его сообщение.

26) Что сделает команда git reset HEAD~1 file.txt? (2 балл)

После выполнения данной команды параметр HEAD~1 будет указывать на предыдущий коммит, что означает, что команда будет использовать информацию о состоянии файла из этого коммита. Однако поскольку в команде git reset не указан параметр, который изменяет состояние рабочего каталога (например, --soft или --hard), изменения в рабочем каталоге не будут затронуты. Таким образом, в результате выполнения команды git reset HEAD~1 file.txt: Изменения в file.txt останутся в рабочем каталоге (они не будут потеряны), но не будут находиться в индексе. Файл file.txt будет отмечен как измененный, и его изменения будут видны с помощью команды git status.

27) Для чего используется команда “git config”? (1 балл)

Команда git config используется для настройки параметров Git. Она позволяет управлять различными аспектами поведения Git на уровне пользователя, проекта или системы. С помощью этой команды можно:

- настройка глобальных параметров

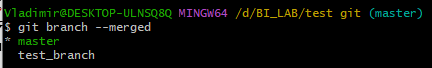
- настройка локальных параметров

- настройка системных параметров

- просмотр настроек

28) Как можно проверить, что ветка уже объединена с “main” веткой? (1 балл)

Чтобы проверить, была ли ветка уже объединена с веткой main, можно использовать команду git branch --merged. Эта команда показывает список веток, которые уже были объединены с текущей веткой, или вы можете указать конкретную ветку, например, main.



29) После того, как вы инициализировали новый Git репозиторий и создали файл “git-quiz.html”, какие из следующих команд не дадут ожидаемый результат при выполнении? (1 балл)

1. git commit -a -m “git quiz web file added”

2. git status   
3. git add .

4. git commit -m "git quiz web file added"

Команды 1. git commit -a -m “git quiz web file added” и

4. git commit -m "git quiz web file added" не дадут ожидаемый результат при выполнении

30) Как очистить вашу директорию от неотслеживаемых (untracked) файлов и поддиректорий – напишите команду. Как очистить директорию от игнорируемых файлов? (2 балла)

git clean -f -d очистит директорию от неотслеживаемых файлов и поддиректорий

git clean -f -X очистит директорию от игнорируемых файлов

31) Как вы можете переименовать удаленный репозиторий? Напишите команду. (1 балл)

git remote rename <old\_name> <new\_name>

32) Что такое "detached HEAD" в Git? Почему оно считается ‘опасным’ состоянием? (2 балла)

В Git "detached HEAD" — это состояние, когда HEAD указывает на конкретный коммит, а не на имя ветки. Это происходит, если выполняется команда git checkout <commit-hash>, не переключаясь на ветку

Это состояние считается опасным, так как может привести к потери изменений. Любые новые коммиты, созданные в этом состоянии, будут привязаны к конкретному коммиту и не связаны с веткой. Если переключиться на другую ветку, эти коммиты могут потеряться.

33\*) Ниже указаны команды. Пожалуйста, напишите сообщения коммитов в порядке, в каком вы увидите их в git log. (4 балла)

Sergei\_Boikov@EPRUPETW0116 MINGW64 ~/Desktop/test (main)

$ touch never\_gonna.txt

Sergei\_Boikov@EPRUPETW0116 MINGW64 ~/Desktop/test (main)

$ git add .

Sergei\_Boikov@EPRUPETW0116 MINGW64 ~/Desktop/test (main)

$ git commit -m "never gonna"

Sergei\_Boikov@EPRUPETW0116 MINGW64 ~/Desktop/test (main)

$ git checkout -b br2

Sergei\_Boikov@EPRUPETW0116 MINGW64 ~/Desktop/test (br2)

$ touch push.txt

Sergei\_Boikov@EPRUPETW0116 MINGW64 ~/Desktop/test (br2)

$ git add .

Sergei\_Boikov@EPRUPETW0116 MINGW64 ~/Desktop/test (br2)

$ git commit -m "push"

Sergei\_Boikov@EPRUPETW0116 MINGW64 ~/Desktop/test (br2)

$ touch you\_down.txt

Sergei\_Boikov@EPRUPETW0116 MINGW64 ~/Desktop/test (br2)

$ git add .

Sergei\_Boikov@EPRUPETW0116 MINGW64 ~/Desktop/test (br2)

$ git commit -m "you\_down"

Sergei\_Boikov@EPRUPETW0116 MINGW64 ~/Desktop/test (br2)

$ git checkout main

Sergei\_Boikov@EPRUPETW0116 MINGW64 ~/Desktop/test (main)

$ git checkout -b br1

Sergei\_Boikov@EPRUPETW0116 MINGW64 ~/Desktop/test (br1)

$ touch git\_you\_up.txt

Sergei\_Boikov@EPRUPETW0116 MINGW64 ~/Desktop/test (br1)

$ git add .

Sergei\_Boikov@EPRUPETW0116 MINGW64 ~/Desktop/test (br1)

$ git commit -m "git you up"

Sergei\_Boikov@EPRUPETW0116 MINGW64 ~/Desktop/test (br1)

$ touch never.txt

Sergei\_Boikov@EPRUPETW0116 MINGW64 ~/Desktop/test (br1)

$ git add .

Sergei\_Boikov@EPRUPETW0116 MINGW64 ~/Desktop/test (br1)

$ git commit -m "never"

Sergei\_Boikov@EPRUPETW0116 MINGW64 ~/Desktop/test (br1)

$ touch gonna.txt

Sergei\_Boikov@EPRUPETW0116 MINGW64 ~/Desktop/test (br1)

$ git add .

Sergei\_Boikov@EPRUPETW0116 MINGW64 ~/Desktop/test (br1)

$ git commit -m "gonna"

Sergei\_Boikov@EPRUPETW0116 MINGW64 ~/Desktop/test (main)

$ git checkout main

Sergei\_Boikov@EPRUPETW0116 MINGW64 ~/Desktop/test (main)

$ git merge br1

Sergei\_Boikov@EPRUPETW0116 MINGW64 ~/Desktop/test (main)

$ git checkout br2

Sergei\_Boikov@EPRUPETW0116 MINGW64 ~/Desktop/test (br2)

$ git rebase main

Sergei\_Boikov@EPRUPETW0116 MINGW64 ~/Desktop/test (br2)

$ git checkout main

Sergei\_Boikov@EPRUPETW0116 MINGW64 ~/Desktop/test (main)

$ git merge br2

**you\_down**

**push**

**gonna**

**never**

**git you up**

**never gonna**

34\*) Какая разница между командами rebase и merge? Когда вы должны делать rebase, а когда merge? (4 балла)

В Git команды rebase и merge используются для интеграции изменений из одной ветки в другую, но они работают по-разному и служат для разных целей:

git merge: слияние создает новый коммит (merge commit), объединяющий изменения из обеих веток. История сохраняет обе ветки, показывая, когда и как они были объединены.

git rebase: «переписывает» историю, перемещая коммиты из одной ветки в другую. Не создает merge-коммит, а применяет коммиты вашей ветки поверх базовой ветки, тем самым делая историю линейной. Полезно для поддержания чистоты истории, особенно когда нет необходимости в сложных разветвлениях.

35\*) Опишите команду git reset. Какие у нее есть опции? Какая разница между командами git reset, git checkout, git revert? Когда вы используете команду git reset, а когда git checkout? (4 балла)

Команда git reset используется для отмены коммитов, перемещения указателя HEAD или отмены индексации изменений.

-- soft: удаляет коммиты, но оставляет изменения в индексе (stage area). Это полезно, если нужно переписать коммиты, но сохранить подготовленные изменения.

-- mixed: удаляет коммиты и сбрасывает изменения из индекса, но оставляет их в рабочем каталоге. Это удобный вариант, если нужно снова проиндексировать изменения перед повторным коммитом.

-- hard: удаляет коммиты и стирает изменения как из индекса, так и из рабочего каталога. Опасная команда, так как изменения полностью удаляются.

git reset: применяется, когда нужно отменить последние коммиты или отменить индексированные изменения в рабочем каталоге. Используется, когда необходимо откатить изменения в локальной ветке без создания нового коммита (например, чтобы переписать историю).

git checkout: используется для переключения между ветками или для просмотра старых коммитов. Полезен при необходимости временно переключиться на другую ветку или коммит без удаления изменений в рабочем каталоге.