4. Git, GitHub

Вопрос 467. Почему прибегают к использованию командной строки?

Ответ: Существуют задачи, которые невозможно разрешить через графические интерфейс (GUI), т.к. он не написан. Поэтому прибегают к использованию командной строки.

Вопрос 468. Чем позволяет управлять командная строка?

Ответ: Командная строка (консоль, терминал) - это программа, которая позволяет управлять компьютером через ввод текстовых команд.

Вопрос 469. Что можно делать из командной строки?

Ответ: просматривать содержимое папок, создавать файлы, удалять файлы, скачивать данные с серверов, устанавливать программы, запускать программы

Вопрос 470. Что такое Git?

Ответ: Git - это система версионного контроля, представляющая собой консольную программу, которая управляется командами через терминал.

Вопрос 471. Что такое репозиторий?

Ответ: Когда вы начинаете работать со своим проектом через Git, папка с проектом становится репозито́рием. В широком смысле репозиторий — это хранилище с данными. Если же говорить о нём в ключе системы контроля версий, того же Git, — это не просто хранилище данных, это хранилище версий файлов проекта.

Вопрос 472. Где хранятся репозитории?

Ответ: Локальные репозитории хранятся на вашем компьютере, а удалённые вне вашего компьютера, например, в облачном сервисе.

Вопрос 473. Что лучше использовать в Windows для работы Git?

Ответ: Для Windows тоже существует аналог Bash, который называется Git Bash.

Вопрос 474. Как в GitBash пометореть где я сейчас нахожусь?

Ответ: Реализуется командой `pwd`

Вопрос 475. Как узнать, что в директории содержится?

Ответ: Реализуется командой ls

Вопрос 476. Как можно через Git Bush управлять файлами?

Ответ: Реализуется через ключи. Их можно узнать, если запросить универсальный ключ `--help`

ls --help

Вопрос 477. Как перемещаться по директории?

Ответ: Для перемещения из одной папки в другую используется команда `cd` (от англ. _change directory —_ «сменить каталог»)_._ Синтаксис у команды такой: `cd имя_папки`.

Если в названии папки есть пробел, то используются кавычки.

Вопрос 478. Как вернуться в предыдущую директорию?

Ответ: Чтобы перейти в директорию выше стаится `cd ..`

Вопрос 479. Как перейти к корень диска, например D:?

Ответ: Это выполнятеся командой `cd /d`

Вопрос 480. Как создать папку в директории?

Ответ: Перейдите через терминал в нужную директорию. Далее введите команду `mkdir` (от англ. _make directory_ — «создать директорию»), передайте ей имя новой папки и нажмите **Enter**:

Вопрос 481. Как создать файл в директории?

Ответ: Введите в терминал команду `touch` (англ. «коснуться») и передайте ей имена файлов с расширением:

Вопрос 482. Как запустить эту программу?

Ответ: Вызвать команду `python proba.py`

Вопрос 483. Как удалять папки и файлы?

Ответ: Для этого передайте команде `rm` (от англ. _remove —_ «удалить») имя файла, для удаления директории `rmdir`, Если директория не пустая - вылезет ошибка, Если все же нужно удалить директорию со всеми файлами надо воспользоваться командой `rm` с ключем `-r`. *Файлы не перемещаются в корзину, а удаляются навсегда

Вопрос 484. Как проще вводит имена папок и файлов, чтобы Git Bush как бы нам подсказывал?

Ответ: Да, можно. Достаточно ввести несколько начальных букв из названия нужной папки и нажать клавишу Tab.

Вопрос 485. Что такое виртуальное окружение Python?

Ответ: Виртуальное окружение - это изолированное окружение среды, которое позволяет нам использовать определенные версии интерпретатора Python и/или библиотек.

Вопрос 486. Как создать виртуальное окружение?

Ответ: Выбрать папку с проектом или папку со скаченным с GitHub репозиторием, В терминале Git Bush в IDE необходимо прописать следующее

`python -m venv venv`

После выполнения этой команды в директории проекта появится папка venv
(от _**v**irtual **env**ironment_, «виртуальное окружение»)

3. Активация виртуального окружения

`source venv/Scripts/activate`

4. Убедиться, что появилось **(venv)**

5. Чтобы деактивировать виртуальное окружение нужно выполнить команду:

`deactivate`

6. Чтобы посмотреть какие библиотеки установлены в виртуальном окружении

`pip list`

Вопрос 487. Что надо сделать, чтобы поделиться своим виртуальным окружением?

Ответ: Надо выполнить команду:

`pip freeze > requirements.txt`

Вопрос 488. Что такое файл requirements.txt?

Ответ: requirements.txt - это файл зависимостей. В этот файл будут построчно записаны имена пакетов и библиотек с указанием их версий: всё, что вы устанавливали в виртуальное окружение проекта.

Вопрос 489. Как удалить виртуальное окружение?

Ответ: `deactivate`

`rm -r venv`

Вопрос 490. Что такое GitHub и для чего он нужен?

Ответ: GitHub - это сайт, где можно завести аккаунт и размещать свой код, совместно работать над собственными или _open source_ проектами (англ. «открытое программное обеспечение»)

Вопрос 491. В чем разница между Git и GitHub?

Ответ: Git и GitHub — не одно и то же. Git — специальная программа, а GitHub — специальный сервис, который поддерживает работу с Git.

_____

Вопрос 492. Что делает Git для GitHub?

Ответ: Git позволяет работать с версионированием твоих проектов локально, на твоём компьютере, а потом передавать всю информацию об изменениях на GitHub.

_____

Вопрос 493. Какие способы аутентификации существуют на GitHub?

Ответ: Аутентифицироваться можно через сетевые протоколы HTTPS или SSH. По этим протоколам передаются данные и происходит безопасное соединение между клиентом и сервером.

_____

Вопрос 494. Что такое SSH?

Ответ: SSH (англ. Secure SHell_ — безопасная оболочка) — это сетевой протокол для зашифрованного соединения между клиентом и сервером; по этому протоколу можно безопасно передавать данные.

_____

Вопрос 495. Что означают параметры Public/Private у репозитория?

Ответ: Public/Private - тип репозитория: публичный доступен всем пользователем интернета, а приватный виден только вам.

_____

Вопрос 496. Что такое README файл в репозитории?

Ответ: Initialize this repository with a README — создать файл README в репозитории. В файле README описывается проект и как с ним работать.

_____

Вопрос 497. Что такое файл .gitignore в репозитории?

Ответ: Add .gitignore — создать .gitignore, файл, где будут перечислены файлы и папки вашего репозитория, которые Git не должен отслеживать и синхронизировать. Укажите пункт Python. Автоматически будет создан гитигнор-файл, стандартно настроенный для любого Python-проекта. В дальнейшем этот файл можно будет отредактировать.

_____

Вопрос 498. Что такое лицензия в репозитории?

Ответ: Add license — добавить лицензию. В лицензии вы устанавливаете права на свой проект. Мы рекомендуем указать лицензию BSD 3 или MIT: они предоставляют хороший баланс прав и ответственности.

_____

Вопрос 499. Из чего состоит файл README?

Ответ: Как правило, README.md состоит из следующего плана:

- Название проекта

- Краткое описание

- Технологии в проекте

- Инструкции по запуску

- Автор

Вопрос 500. Как клонировать репозиторий на локальный компьютер?

Ответ: git clone git@github.com:IvanZaycev0717/**********.git

5. Переходим в эту папку

6. Создаём виртуальное окружение

`python -m venv venv`

7. Активируем виртуальное окружение

`source venv/Scripts/activate`

8. Выполняем команду

`pip install -r requirements.txt`

Вопрос 501. Перечислите статусы файлов Git?

Ответ: **Неотслеживаемый** (англ. _untracked_).

2. **Отслеживаемый,** _staged_, добавленный в _Staging Area_ (англ. «плацдарм», «временное хранилище»). Иначе в Git это называют «добавить в индекс».

3. **Изменённый** (англ. _modified_).

4. **Боевой**, на жаргоне разработчиков «закоммиченный» (англ. _committed_, «брошенный в бой»). [В документации](https://git-scm.com/book/en/v2/Git-Basics-Recording-Changes-to-the-Repository) такие файлы называются **неизменёнными** (от англ. _unmodified_). Иными словами, это файлы, в которых не было никаких изменений с момента последнего коммита (с момента, как разработчик «бросил их в бой»).

Вопрос 502. Как отследить статус файлов в репозитории?

Ответ: Выполнить команду:

`git status`

Это говорит о том, что ничего изменено не было. И сейчас актуальные версии всех файлов.

Вопрос 503. Если файлы были изменены как добавить к локальный репозиторий индекс?

Ответ: `git add название_файла`

Если нужны все репозитории файла, что намного удобнее, используют `git add .`

Вопрос 504. Как файл с индексом можно перевести в состояние "неотслеживаемый»?

Ответ: git rm --cached <file>

Вопрос 505. Как зафиксировать изменения в файле, "бросить его в бой" – т.е. сделать коммит?

Ответ: git commit -m 'Что собой представляет'`

Ключ `-m` (от англ. _message_, «послание») даёт возможность добавить к коммиту комментарий, который пишется в кавычках после ключа.

В комментариях описывайте, какие изменения были сделаны в коммите, иначе через неделю вы уже не вспомните, что и зачем было написано. Git понимает кириллицу, но комментировать по-английски — хороший тон.

Вопрос 506. Как отправить файлы на сервер?

Ответ: `git push`

Вопрос 507. Что делать, если в удаленный репозиторий попали лишние файлы и папки?

Ответ: `git rm --cached < название файла >`

`git commit -m "Deleted new file"`

`git push`

Вопрос 508. Как посмотреть все коммиты, которые были сделаны?

Ответ: git log

Вопрос 509. Как сделать откат к определенному коммиту?

Ответ: Можно откатиться на один коммит назад в определённом файле, указав его имя через HEAD: git reset HEAD program.py

---

Вопрос 510. Кто такой техлид проекта?

Ответ: В IT командах техлид определяет стек технологий для проекта, отвечает на сложные технические вопросы и отслеживает прогресс команды.

Вопрос 511. Кто такой тимлид?

Ответ: Тимлид — это один из разработчиков; его задача — помогать другим участникам работать в команде, инициировать обсуждения по важным вопросам, отвечать за конечный результат.

Вопрос 512. Что такое таск-трекеры?

Ответ: Таск-трекеры — это системы, наглядно отображающие актуальное состояние процесса. Их применяют для координации работы и отслеживания статуса выполнения задач в пределах проекта.

Вопрос 513. Что такое встреча daily?

Ответ: daily — ежедневный отчёт о проделанной работе. В специальной таблице тимлид будет отмечать, какие задачи выполнены к моменту отчёта.

Вопрос 514. Что такое встреча демо?

Ответ: демо — общая встреча о выполненных задачах, обсуждения сложностей и возникших вопросов. Эту встречу будет проводить тимлид.

Вопрос 515. Что такое встреча ретро?

Ответ: ретро — встреча, на которой команда анализирует свой опыт совместной работы, оценивает, что получилось, а что в следующих проектах можно сделать лучше.

Вопрос 516. Что в первую очередь должен сделать тимлид при совместной разработке?

Ответ: Первое, что должен сделать тимлид — добавить соразработчиков в свой репозиторий.

В своём репозитории на GitHub тимлид должен перейти во вкладку _Settings_, затем в левом боковом меню выбрать _Collaborators_ и на открывшейся странице нажать кнопку _Add people_. В открывшемся окне необходимо указать данные разработчика, которому предоставляется доступ к репозиторию в качестве соразработчика: его никнейм, полное имя или адрес электронной почты пользователя GitHub. Если пользователь найден, то его нужно выбрать и нажать на кнопку _Select a collaborator above_; после этого ваш коллега получит приглашение в репозиторий, оно придёт на адрес электронной почты, привязанный к GitHub. Соразработчики должны принять приглашение, перейдя по ссылке в письме. Ссылка будет активна 7 дней; если за это время приглашенный не воспользуется ссылкой-приглашением, то эту процедуру придётся повторить. Когда все примут приглашение — репозиторий тимлида будет готов к совместной разработке.

Вопрос 517. Что такое ветка при командной разработки?

Ответ: Ветка — это изолированный поток разработки, в котором можно делать коммиты так, что они не повлияют на код в других ветках проекта.

Вопрос 518. Из чего состоит репозиторий при командной разработке?

Ответ: Основная ветка репозитория называется master или main. Эта ветка создаётся автоматически, когда в проекте инициализируется Git и создаётся первый коммит. В репозитории тимлида сейчас должна быть только эта ветка. Разработчики проекта могут создавать новые ветки и переключаться между существующими. Обычно в ветке master хранят «продуктовую» версию кода — отлаженную и работающую; разработку же ведут в другой, отдельной ветке, например в develop. Это хорошая практика, будем придерживаться именно её.

Создать новую ветку можно прямо в репозитории на GitHub, но чаще всего разработчики делают это локально, в склонированном репозитории; в вашей команде создать ветку **develop** должен тимлид.

Вопрос 519. Как можно создать свою ветку в проекте?

Ответ: Создать ветку можно командой `git branch название_ветки`. При выборе имени помните, что имя ветки не должно содержать пробелов: это вызовет ошибку.

Но ветка **develop** создана и сохранена лишь в локальном репозитории тимлида, а в репозитории на GitHub её пока нет. Чтобы сделать её доступной для других разработчиков, тимлид должен выполнить команду:

`git push --set-upstream origin develop`

Вопрос 520. Что надо сделать прежде чем написать новую часть проекта?

Ответ: Прежде чем начать писать новую часть проекта или так называемую “фичу” (от англ. feature), для неё создают отдельную ветку в Git. При этом в качестве источника выступает ветка develop.

Вопрос 521. Как переключаться между ветками в проекте?

Ответ: git switch <название ветки>

Вопрос 522. Какое название подойдет ветки, если там исправление багов?

Ответ: Если в ветке планируется ловля и исправление багов, ей подойдёт название, начинающееся с `bugfix`:

Вопрос 523. Что надо сделать чтобы ветка стала видна и доступна для других разработчиков ?

Чтобы новая ветка стала видна и доступна для других разработчиков, создатель ветки должен синхронизировать изменения с удалённым репозиторием при помощи команды

`git push --set-upstream origin <имя ветки>`

Вопрос 524. Что делать другим разработчкам, чтобы иметь актуальное состояние репозитория?

Ответ: А другие разработчики, чтобы иметь актуальное состояние репозитория, должны выполнить команду `git fetch`.

Вопрос 525. Как производится промежуточное код-ревью перед слиянием веток?

Ответ: Когда работа над конкретной фичей завершена, то прежде чем смержить ветку фичи с веткой **develop** — стоит провести **код-ревью**. Эта возможность доступна на GitHub через механизм **pull request**.

Вопрос 526. Что такое Pull request?

Ответ: **Pull request** (PR) — это способ известить команду о готовности определённых изменений в коде с возможностью отревьюить новый код.

_____

Вопрос 527. Кто участвует в Pull request?

Ответ: Одновременно с запросом автор кода может запросить ревью, указав конкретных участников команды в качестве ревьюеров.

_____

Вопрос 528. Что делать с замечаниями в Pull request?

Ответ: оставить общий комментарий к pull request, не принимая никакого решения; оставить общий комментарий к pull request, одобрив предлагаемый код; отправить автору кода комментарии и попросить доработать код.

_____

Вопрос 529. Что надо делать прохождения Pull request?

Ответ: После решения задачи вашу изолированную ветку нужно объединить с веткой **develop**, залить в неё результаты вашей работы. А когда вся работа над проектом будет завершена, рабочую ветку **develop** тимлид «вольёт» в главную и отправит на проверку ревьюеру. Этот процесс называется «слияние веток», или **merge** («мёрдж», «мёрж», «мердж», «мерж» — как только ни произносят это слово в русскоязычных командах).

_____

Вопрос 530. Какая команда делает мердж?

Ответ: `git merge feature/email-validation`

_____

Вопрос 531. Что может возникнуть при попытки смержить файлы?

Ответ: Иногда при слиянии веток могут возникнуть конфликты: например, два разработчика в разных ветках изменили код в одной и той же строке, и в процессе слияния веток Git не может решить, какой код оставить в финальной версии.

`git merge readme-fix2`

Git сообщает о конфликтах:

В коде файлов с конфликтами Git даёт подсказки на тех строках, где в разных ветках текст отличается.

Эту процедуру повторяют для всех конфликтующих файлов. После этого делается коммит, чтобы зафиксировать изменения — и ещё раз синхронизация с уделённым репозиторием. Теперь слияние веток должно получиться! Увидеть потенциальные конфликты можно и иначе, например уже на этапе создания pull request. GitHub предупредит о возможных конфликтах, но сделать pull request всё равно не запретит. А сами конфликты можно будет решить через интерфейс GitHub, отредактировав конфликтующий код прямо на сайте, после нажатия на кнопку _Resolve conflicts_.

После окончания редактирования нужно пометить конфликт, как решённый, а также сделать коммит.