Методичка к Блоку 2. «Управление кодовой базой (GIT)»

Список программного обеспечения

Название ПО	Ссылка	Примечание
GIT	https://git-scm.com/downloads	Использовать актуальную версию
Visual Studio Code (VS Code)	https://code.visualstudio.com/download	Популярный современный редактор кода, поддерживающий множество языков программирования, в том числе 1С
GitExtensions	http://gitextensions.github.io/	Хороший графический клиент для GIT, использовать актуальную версию
vanessa-bootstrap	https://github.com/vanessa-opensource/vanessa-bootstrap	Шаблон проекта, использовать актуальную версию
OneScript	https://oscript.io/downloads	Использовать актуальную версию
GitSync	https://github.com/oscript-library/gitsync	Инструмент синхронизации Хранилища с репозиториями GIT, устанавливать через opm командой `opm install gitsync`

Краткая справка по командам GIT

Описание	Пример команды
Первоначальные настройки (выполняются разово)	
Установка имени и email'а в глобальных настройках git'а от имени которых будут совершаться операции (эти имя и email будут в комментариях к коммиту).	`git config --global user.name "Ваше имя"` `git config --global user.email Ваш@email`
Настройка для корректного вывода кириллицы в путях (в именах файлов и каталогов)	`git config --global core.quotePath false`
Важная настройка для корректного преобразования переводов строк при коммитах, когда репозиторий находится под одной ОС (GitLab, установленный в Linux), а работаете вы под другой (Windows).	`git config --global core.autocrlf true` `git config --global core.safecrlf false`
Настройка для увеличения буфера. Нужна, когда подключаете репозитории по HTTP, при этом коммитите очень большие файлы	`git config --global http.postBuffer 1048576000`
Включение системной поддержки длинных путей. Важно! Эту команду выполнять в консоли, запущенной от имени администратора системы!	`git config --system core.longpaths true`

Задание алиасов (псевдонимов) командам, чтобы можно было использовать сокращенные названи команд, например, чтобы вместо git commit можно было писать в консоли git ci
 	
См. подробнее в https://git-scm.com/book/ru/v2/Основы-Git-Псевдонимы-Git
	`git config --global alias.co checkout`
	`git config --global alias.br branch`
	`git config --global alias.ci commit`
	`git config --global alias.st status`

Вывести все текущие значения настроек GIT	`git config -l --show-origin`
Создание репозитория	

Создание нового локального репозитория в каталоге my_repo	`mkdir my_repo`
	`cd my_repo`
	`git init`

Клонирование внешнего репозитория. Будет создана папка repo и в нее клонирован репозиторий	`git clone https://mygit.com/repo.git`

Клонирование внешнего репозитория в текущую пустую папку	`mkdir repo`
	`cd repo`
	`git clone https://mygit.com/repo.git -`

Клонировать от имени другого пользователя	`git clone https://ТВОЙ_ЛОГИН@github.com/путь/к/репо.git`
Регулярное использование	
Узнать текущее состояние репозитория	`git status`
Добавить новый или измененный файл проекта в индекс репозитория	`git add ИмяФайла`
Добавить все измененные/новые файлы из каталога ИмяКаталога в индекс (рекурсивно, включая файлы в подкаталогах)	`git add ИмяКаталога`
Добавить все измененные/новые файлы из текущего каталога рекурсивно в индекс репозитория	`git add .`
Зафиксировать *все* помещенные в индекс репозитория изменения. Для ввода сообщения к коммиту откроется текстовый редактор, указанный в настройке гита с именем `core.editor`	`git commit`
Зафиксировать изменения конкретного файла, помещенные в индекс. Для ввода сообщения к коммиту откроется текстовый редактор, указанный в настройке гита с именем `core.editor`	`git commit ИмяФайла`
Зафиксировать все изменения в репозитории, помещенные в индекс с указанным сообщением к коммиту	`git commit -m "Текст сообщения к коммиту"`
Добавить все измененные файлы в текущем каталоге в индекс и сразу зафиксировать в репозитории с указанным сообщением к коммиту. Важно! Новые файлы автоматически этой командой не добавятся в индекс. Для добавления новых файлов обязательно	`git commit -a -m "Текст сообщения к коммиту"`

использовать `git add`	
Вывести историю изменений	`git log`
Вывести историю изменений в компактном виде	`git log --oneline`
Вывести список всех существующих веток репозитория	`git branch -l`
Создать новую ветку от текущей	`git branch ИмяВетки`

Переключиться на существующую ветку	`git checkout ИмяВетки`
	или
	`git switch ИмяВетки`

Создать новую ветку от текущей ветки и сразу переключиться на нее	`git checkout -b ИмяВетки`
	или
	`git switch -c ИмяВетки`

Удалить ветку с проверкой, была ли изменения из этой ветки смержены куда-либо. Если ветка не была смержена, удаления не будет (будет выведено сообщение об этом)	`git branch -d ИмяВетки`
Безусловное удаление ветки (независимо от того, была смержена или нет)	`git branch -D ИмяВетки`
Смержить (слить, сравнить/объединить) две ветки. Ветка, указанная в качестве аргумента сливается в текущую ветку.	`git merge ИмяВетки`
Отменить начатую операцию слияния	`git merge --abort`
Работа с внешним (удаленным) репозиторием	
Вывести список подключенных внешних репозиториев	`git remote -v`
Получить из внешнего репозитория все изменения, но не применять их к текущей ветке	`git fetch`
Получить из внешнего репозитория все изменения и сразу применить к текущей ветке	`git pull`
Поместить изменения из	`git push`

текущей ветки в удаленный репозиторий	

Как клонировать репозиторий, и по с опубликованный на GitHub

Что мы будем делать?

Для работы с удаленными репозиториями GIT поддерживает два протокола: HTTPS и SSH.

Сервер управления репозиториями GitHub поддерживает оба протокола. Оба из них требуют авторизацию: HTTPS — по логину и паролю, а SSH — по ключу.

До недавнего времени GitHub в качестве пароля при доступе по HTTPS использовал основной пароль, под которым вы авторизуетесь в веб-интерфейсе GitHub'а.

Но с 31 августа 2021 года авторизация по паролю при использовании протокола HTTPS в GIT-клиентах была удалена.

Теперь вместо основного пароля нужно использовать временный персональный токен (Personal access token), который нужно предварительно сгенерировать.

В данной инструкции объясняется, как создать такой токен и как его использовать при операциях с репозиториями, опубликованными на GitHub'е.

Краткая инструкция

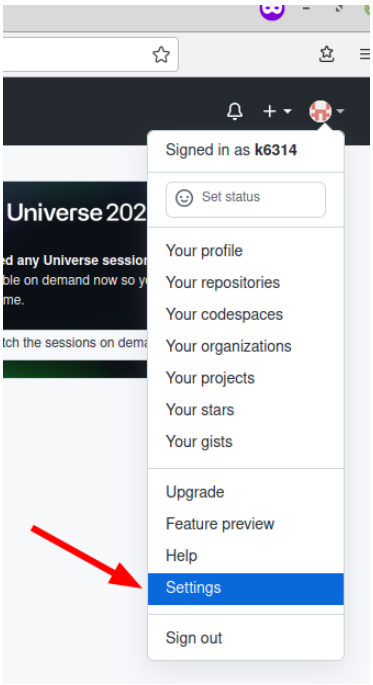
1. Создать токен:
    a. авторизоваться в веб-интерфейсе GitHub
    b. перейти на страницу https://github.com/settings/tokens (меню в правом верхнем углу — Settings — Developer settings — Personal access tokens)
    c. нажать кнопку Generate new token
    d. обязательно скопируйте полученный токен и сохраните его, после того, как уйдете со странице, токен скроется и больше не будет возможности его увидеть
2. Как использовать токен (пример):

```
git clone https://github.com/infostart-devops/devops2023.git
Cloning into 'devops2023'...
Username for 'https://github.com': <ВВОДИМ ВАШ ЛОГИН НА GITHUB>
Password for 'https://ВАШ_ЛОГИН_НА_GITHUB@github.com': <ВВОДИМ ТОКЕН>
```

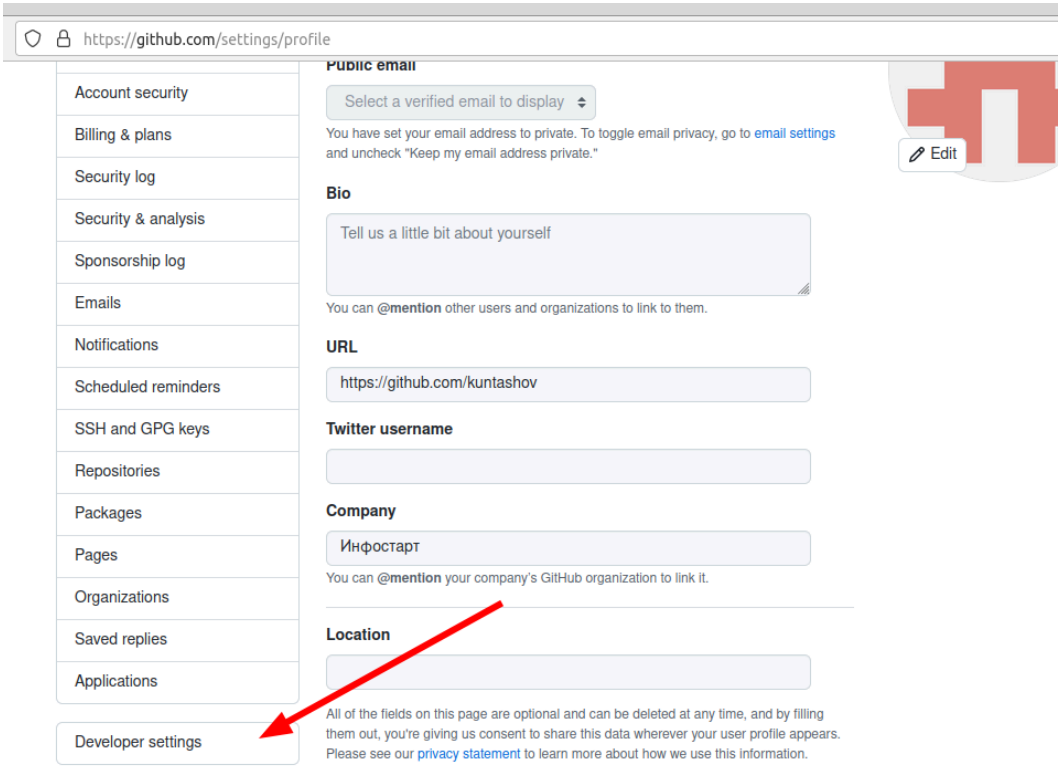
Подробная инструкция (в картинках)

Создание персонального токена

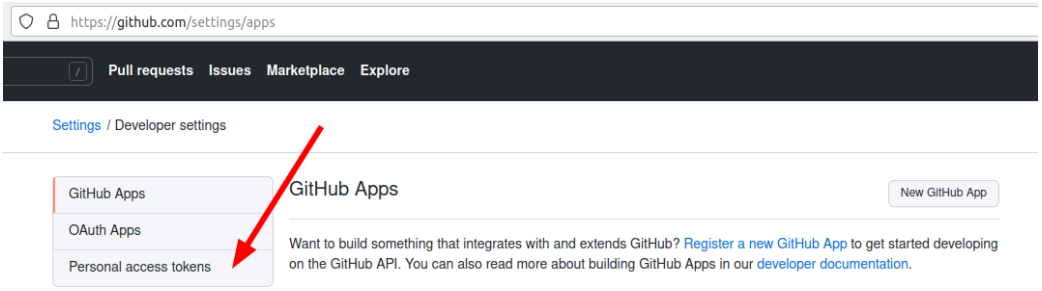
Авторизуйтесь на GitHub, после чего перейдите на страницу настроек:



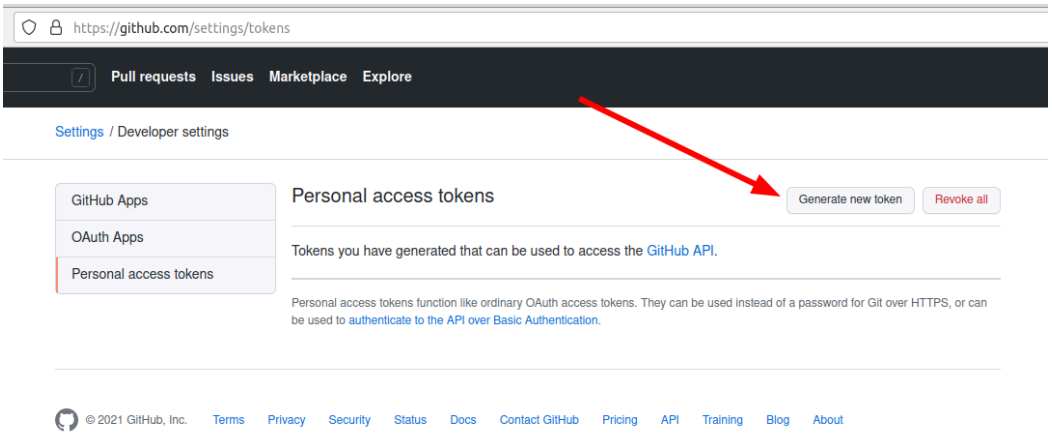
В левом меню выберите пункт “Developers settings”:



Откройте таб «Personal access token»:

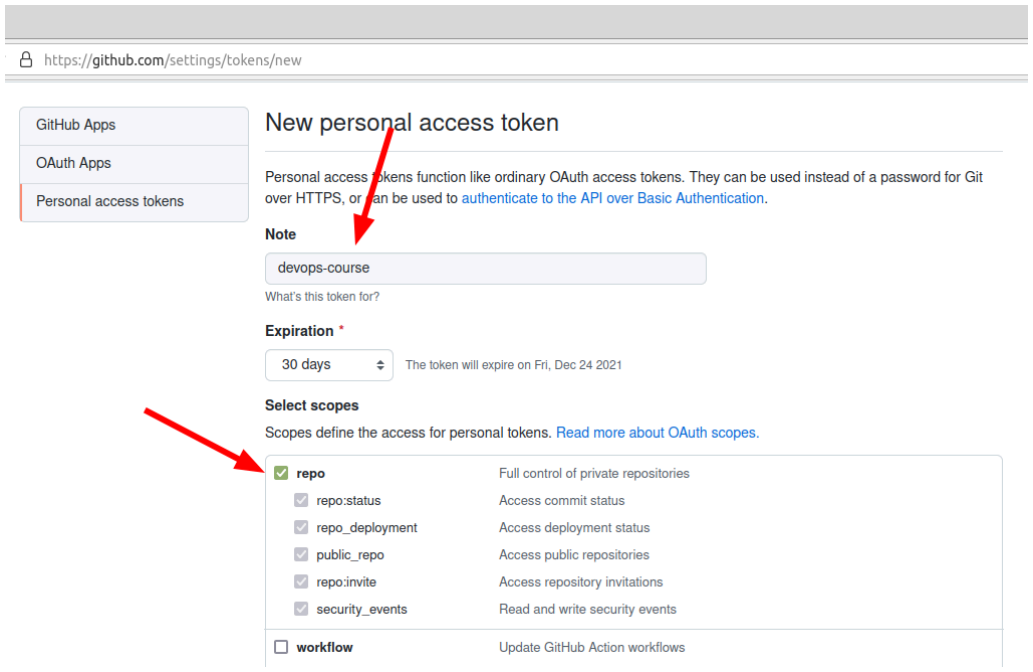


Нажмите кнопку «Generate new token»:

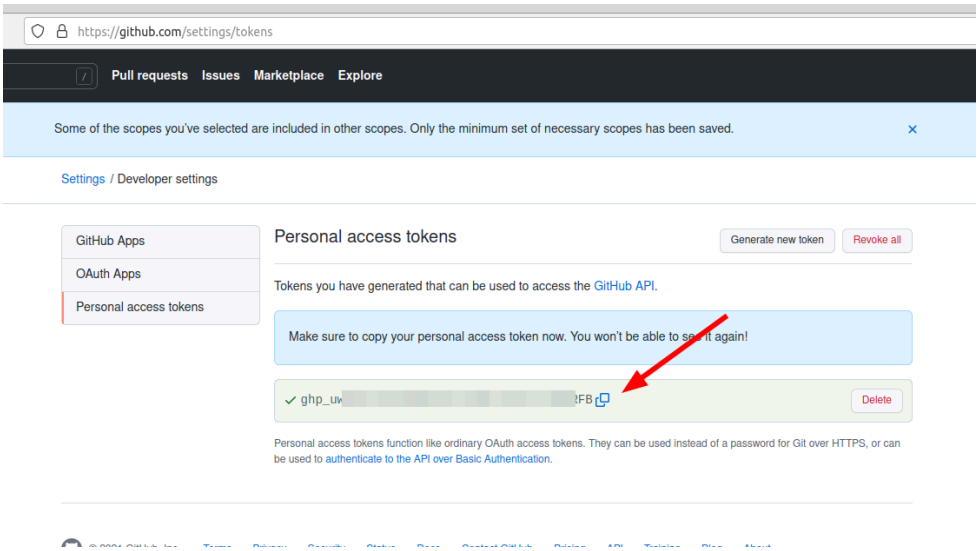


Заполните обязательные свойства токена:

- Наименование (на скриншоте devops-course)
- В группе флагов «Select scopes» отметьте флаг **repo** и сохраните токен (кнопка «Generate token» внизу формы создания токена):



В результате вы увидите страницу с новым токеном:



Важно! Обязательно сразу скопируйте токен, т.к. после закрытия страницы он будет скрыт и посмотреть его не будет возможности.

Если вы не сохранили токен, то просто удалите запись о нем и сгенерируйте новый.

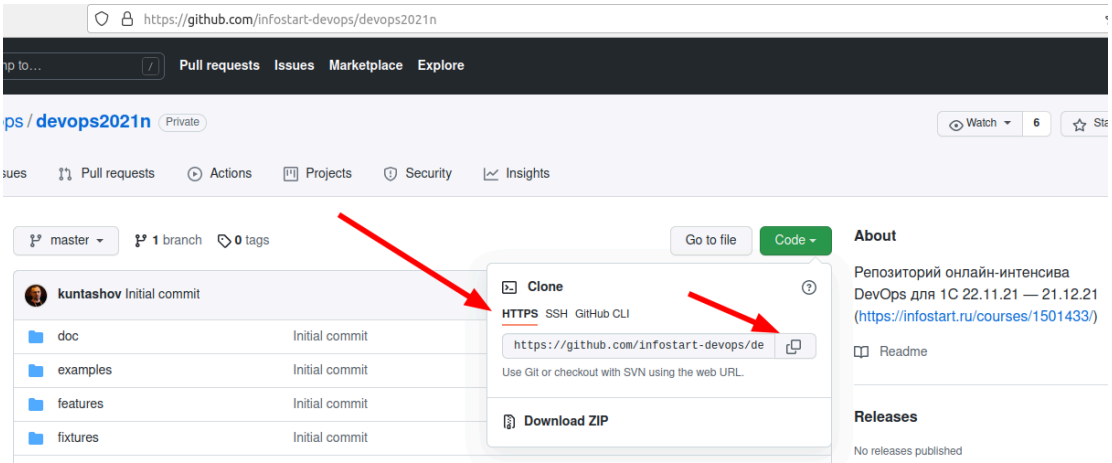
Также обратите внимание, что у токена есть “срок годности” — по умолчанию 30 дней. По истечении этого срока токен перестает работать и нужно создать новый.

Как использовать токен (на примере клонирования репозитория)

Токен использовать просто — в любом git-клиенте он заменяет пароль пользователя.

Например, нам нужно склонировать какой-либо из приватных репозиториев на гитхабе.

Во-первых, идем в этот репозиторий, нажимаем в правом верхнем углу зеленую кнопку “Code” и копируем оттуда со страницы HTTPS адрес репозитория:



Затем, в консоли выполняем команду git clone и в качестве аргумента указываем скопированный адрес:

```
https://github.com/infostart-devops/devops2024o.git
Cloning into 'devops2024o'
Username for 'https://github.com': <ВВОДИМ ВАШ ЛОГИН НА GITHUB>
Password for 'https://ВАШ_ЛОГИН_НА_GITHUB@github.com': <ВВОДИМ ТОКЕН>
```

На запрос имени пользователя (username) указываем свой логин на гитхабе, а на запрос пароля (password) — указываем скопированный токен.

Gitsync и клиент-серверная база

По умолчанию для своей работы gitsync создает и использует временную файловую базу данных.

Но на компьютере, где разворачивается gitsync, может не быть возможности использовать файловые базы: самый частый случай: не установлены/не доступны лицензии для файловой версии ни аппаратные, ни программные.

В этом случае можно использовать служебную клиент-серверную базу, явно указав ее gitsync'у.

Такую базу нужно предварительно создать в кластере 1С.

Затем, чтобы gitsync эту клиент-серверную базу использовать в качестве служебной необходимо указать соединение, задав в командной строке gitsync опцию **`-C <АдресБазы>`** (синонимы: **`--ibconnection <АдресБазы>`**, **`--ib-connection <АдресБазы>`**), где **`<АдресБазы>`** — строка соединения с базой в формате **`/SИмяСервера\ИмяБазы`** (обратите внимание на **`/S`**).

Пример:

```
gitsync -C /SИмяСервера\ИмяБазы - U ИмяПользователя -P Пароль init
```

Также можно указать строку соединения в переменных окружения **`GITSYNC_IB_CONNECTION`** или **`GITSYNC_IBCONNECTION`** (синонимы) непосредственно перед выполнением команды gitsync или глобально.

См. также комментарии в статье https://infostart.ru/1c/articles/1157400/

Дополнительные материалы

Описание	Ссылка
ЧаВо (FAQ) онлайн-интенсива «DevOps для 1С» — список часто задаваемых слушателями курса вопросов	https://itlab.infostart.ru/devops-course-faq
Официальная, исчерпывающая справка/самоучитель по GIT	https://git-scm.com/book/ru/v2/
«Правила полета на GIT» — разбор большого количества кейсов по использованию GIT в примерах, например, “Что делать, если я не правильно написал комментарий к коммиту?” и т.п.	https://github.com/k88hudson/git-flight-rules/blob/master/README_ru.md
Описание организации процесса GitFlow	https://www.atlassian.com/ru/git/tutorials/comparing-workflows/gitflow-workflow
Описание процесса GitHub Flow	https://habr.com/en/post/346066/
Описание процесса GitLab Flow	https://habr.com/en/company/softmart/blog/316686/
Технология разветвленной разработки от 1С	https://its.1c.ru/db/v8std/content/709/hdoc
Интерактивный самоучитель по работе с ветками в GIT	https://learngitbranching.js.org/