СИСТЕМА УПРАВЛЕНИЯ ВЕРСИЯМИ

2023

СОДЕРЖАНИЕ

ВВЕДЕНИЕ
ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ
КЛАССИФИКАЦИЯ СИСТЕМ
ФОРМАТЫ ХРАНИМЫХ ДОКУМЕНТОВ
ПОРЯДОК РАБОТЫ
ГРАФ ИЗМЕНЕНИЙ
GIT
GIT: УСТАНОВКА
TORTOISE GIT
TORTOISE GIT: УСТАНОВКА
TORTOISE GIT: СОЗДАНИЕ РЕПОЗИТОРИЯ
TORTOISE GIT: ИЗВЛЕЧЕНИЕ РАБОЧЕЙ КОПИИ
TORTOISE GIT: ФИКСИРОВАНИЕ ИЗМЕНЕНИЙ
TORTOISE GIT: СИНХРОНИЗАЦИЯ С РЕПОЗИТОРИЕМ
РЕЗЕРВИРОВАНИЕ И ДУБЛИРОВАНИЕ

ВВЕДЕНИЕ

Ситуация, когда электронный документ за время своего существования претерпевает ряд изменений, достаточно типичная. При этом бывает важно иметь не только последнюю версию, но и несколько предыдущих. В самом простом случае можно хранить несколько вариантов документа, нумеруя их соответствующим образом. Такой способ неэффективен, так как: приходится хранить несколько практически идентичных копий документа, требуется повышенное внимание и дисциплина — все это часто ведет к ошибкам и потере данных. Поэтому, были разработаны специальные средства для автоматизации подобной работы.

Система управления версиями или ***Система контроля версий***

- Version Control System, VCS, Revision Control System
- Программное обеспечение для облегчения работы с изменяющейся информацией.
- Позволяет хранить несколько версий одного и того же документа.
- Позволяет при необходимости извлечь:
  - любую версию,
  - любую ревизию (изменение) для версии.

Изменения хранятся в специальном хранилище (***репозиторий***).

Области применения:

- При разработке программного обеспечения для хранения исходных кодов проектов:
  - встраиваемые системы,
  - прикладные системы,
  - управляющие программы ПЛК и проекты SCADA.
- При разработке различных электронных документов:
  - конструкторская,
  - технологическая (в том числе технологические карты, файлы настроек и т. п.).

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

Версия — это символическая метка (имя) ответвления - директория, хранящая историю изменений ветви или ствола (например, релиз-версия — tags/1.0.0, рабочая версия — branches/1.0.x).

Ветвь — независимое направление (копия, версия) проекта. Документы в разных ветвях имеют одинаковую историю до точки ветвления (откуда было создано ответвление) и разные — после нее. Ветвь может отходить от другой ветви, но корнем является — ствол.

Граф — графическое представление ветвления проекта.

Документ (файл) — минимальная единица данных, хранимая в репозитории.

Конфликт — ситуация, когда несколько пользователей сделали изменения одного и того же участка документа (фрагмента), один пользователь зафиксировал в репозитории свои изменения, а второй пытается зафиксировать свои позже. В данном случае возникает конфликт «интересов» - система контроля версий не может понять чьи изменения принять за достоверные. По-умолчанию, достоверными считаются данные пользователя, который первым зафиксировал свои изменения. Пользователю, которому был выпал конфликт необходимо его разрешить (если у него есть такие полномочия) — т. е. разобраться и принятиь решение чти данные принять за достоверные.

Ревизия (изменение) — номер изменения, который определяется специальным счетчиком. Счетчик изменений — один на весь репозиторий. Т.е. это число характеризует количество зафиксированных изменений, сделанных в пределах всего репозитория.

Репозиторий (ханилище) — место, где система управления версиями хранит все документы проекта вместе с историей их изменения. Под историей изменения понимается: изменение (копия или фрагмент документа) и дополнительная информация (имя автора изменения, дата и время изменения, комментарий).

Ствол — основная ветвь (версия, main) проекта. Политика работы со стволом может оличаться от проекта к проекту, но в целом она такова: большинство изменений вносится в ствол, но если требуется серьезное изменение, способное привести к нестабильности, то создается ветвь, которая позже, при достижении нужной стабильности, сливается со стволом.

Фрагмент — измененная часть документа.

commit — процедура фиксации изменения.

main — ствол, основная версия проекта.

branches — директория репозитория, в которой располагаются рабочие версии проекта.

tags — директория репозитория, в которой располагаются релиз-версии проекта.

КЛАССИФИКАЦИЯ СИСТЕМ

Системы управления версиями классифицируют
- По модели управления:
  - локальные (local only)
  - сетевые централизованные, клиент-серверные (client-server)
  - сетевые децентрализованные, распределенные, клиент-серверные (distributed)
- По модели лицензирования:
  - свободные / открытый код (free / open-source)
  - коммерческие / проприетарные (commercial / proprietary)

Локальные

Это системы контроля версий 1-го поколения:
- Репозиторий:
  - располагается на локальном ПК:
    - непосредственно там, где ведется работа с проектом;
  - является основным (централизованный), когда:
    - фиксация изменений, ветвлений, слияний — только в репозиторий.
- Режим доступа:
  - однопользовательский.

Сетевые централизованные, клиент-серверные

Это системы контроля версий 2-го поколения:
- Репозиторий:
  - является основным (централизованным):
    - когда изменения, ветвления, слияния — фиксируются только в нем;
  - располагается на сетевом ПК (сервере):
    - подключен к локальной сети или сети Интернет
      (система контроля верский работает в режиме сервера);
  - может располагаться на локальном ПК:
    - непосредственно там, где ведется работа с проектом,
    - без подключения к какой либо сети
      (система контроля версий работает в локальном режиме).
- Режим доступа:
  - однопользовательский (система контроля версий работает локальный режиме);
  - многопользовательский (система контроля версий работате в режиме сервера).

Сетевые децентрализованные, распределенные, клиент-серверные

Это системы контроля версий 3-го поколения.
Аналог сетевой централизованной модели, но репозиторий здесь не является основным:
- Рабочая копия сама по себе является репозиторием:
  - когда изменения, ветвления, слияния — сначала фиксируются в этой рабочей копии,
    а далее они синхронизируются с основным репозиторием (если он имеется).
- Основной репозитории может вообще отсутствовать:
  - в данном случае модель системы становится полностью локальной.

КЛАССИФИКАЦИЯ СИСТЕМ

Модель управления	Модель лицензирования	Система (год появления)
1-е поколение **локальные** (local only)	свободные / открытый код (free / open-source)	SCCS (1973) RCS (1982)
	коммерческие / проприетарные (commercial / proprietary)	The Librarian (1969) Panvalet (1970) PVCS (1985) QVCS (1991)
2-е поколение **сетевые централизованные, клиент-серверные** (client-server)	свободные / открытый код (free / open-source)	**CVS** (1986) CVSNT (1998) QVCS Enterprise (1998) **Subversion** (2000)
	коммерческие / проприетарные (commercial / proprietary)	Software Change Manager (1970) Dimensions CM (1980) SCLM (1980) DSEE (1984) Synergy (1990) ClearCase (1992) CMVC (1994) Visual SourceSafe (1994) Perforce Helix (1995) StarTeam (1995) Integrity (2001) AccuRev SCM (2002) Surround SCM (2002) Vault (2003) **Azure** DevOps Server (2005) Team Concert (2008) Services (2014)
3-е поколение **сетевые децентрализованные, распределенные** (distributed)	свободные / открытый код (free / open-source)	Code Co-op (1997) BitKeeper (2000) **GNU arch** (2001) Darcs (2002) DCVS (2002) ArX (2003) Monotone (2003) **GNU Bazaar** (2005) **Mercurial** (2005) **Git** (2005) **Fossil** (2007) Breezy (2017)
	коммерческие / проприетарные (commercial / proprietary)	TeamWare (1992) Plastic SCM (2006) **Azure** DevOps Server (2013) Services (2014)

ФОРМАТЫ ХРАНИМЫХ ДОКУМЕНТОВ

В первую очередь:

- документы (файлы) текстовой нотации
  txt, xml, html, css, json, hex, исходный код (c, cpp, h, js, py, php) и т. п.

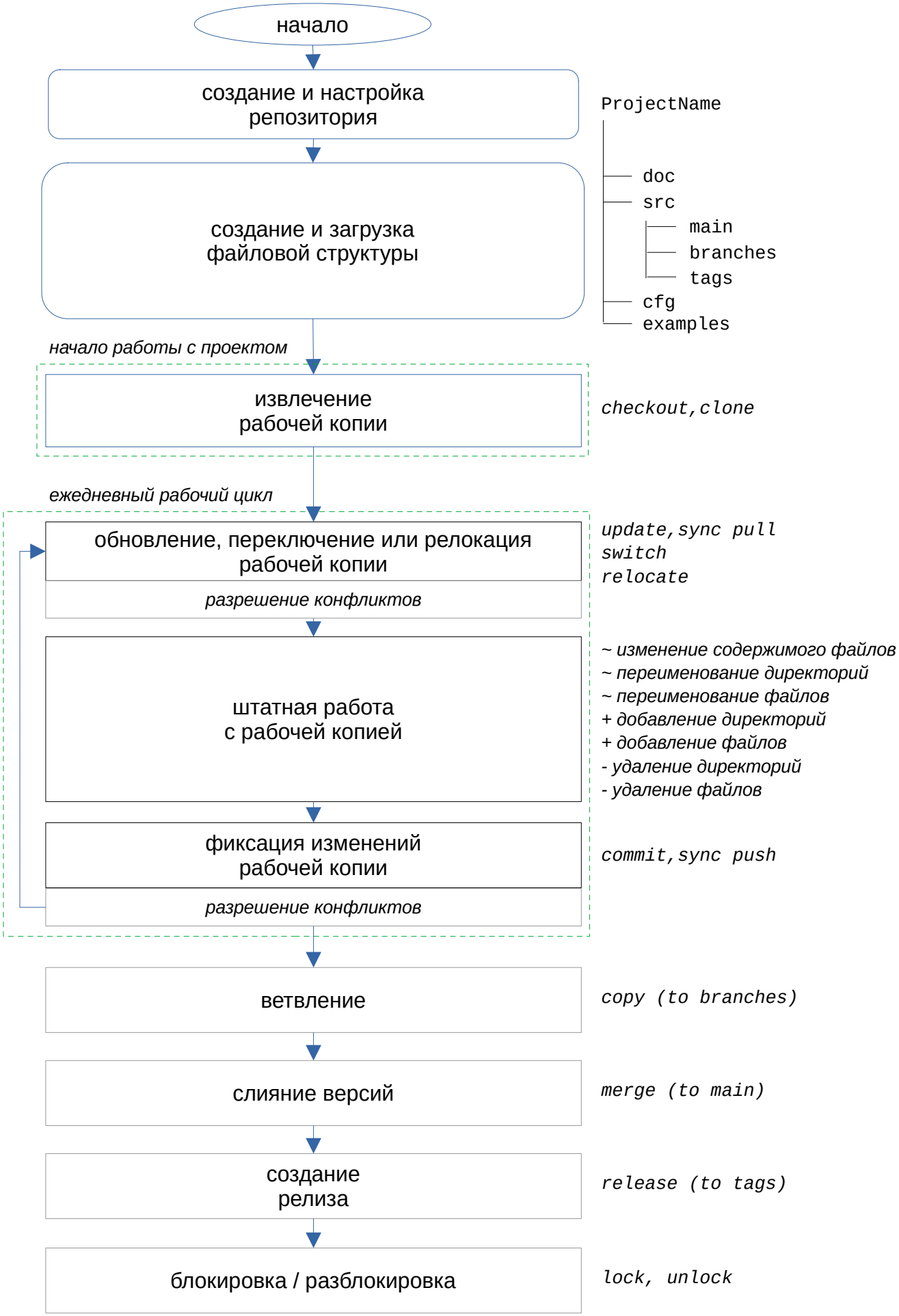
Во вторую очередь (зависит от размера):

- документы (файлы) пакетной нотации
  doc, docx, xls, xlsx, pdf и т. п.

Далее (редко, все зависит от размера)

- объектные (jar, obj), исполняемые (exe, bin, elf), архивы, изображения, видео
  эти документы, из-за их большого размера, обычно хранят вне репозитория (например, в облачном хранилище, а в репозиторий помещают ссылки на них)

ПОРЯДОК РАБОТЫ



ПОРЯДОК РАБОТЫ

Приведенный порядок работы — является набором часто используемых действий и не является каким либо правилом, требованием, стандартом.

Создание и настройка репозитория

Создается пустой репозиторий и задаются его настройки
- имя
- права доступа / роли пользователей
- …

Обычно, на один репозиторий создается на один проект.

Бывает так, что один репозиторий создается для группы проектов — это целесообразно, когда есть несколько небольших проектов, которые можно объединить под одним общим названием (в данном случае каждый небольшой проект является по сути подпроектом).

Создание и загрузка файловой структуры

Создаются директории для удобной группировки хранимых документов (файлов)

Для проектов программных продуктов рекомендуется следующая структура:
- doc (проектная и технологическая документация, руководства);
- src (исходный код):
  - main,
  - branches,
  - tags;
- cfg (файлы настроек, конфигурации);
- examples (примеры).

Начало работы с проектом

Извлечение рабочей копии

Действие выполняется, обычно, единожны — перед началом работы с проектом.

Рабочая копия извлекается с помощью специальной команды:
- задается цель извлечения (определенная директория проекта или весь проект);
- задается номер извлекаемой версии;
- задается номер ревизии (изменения) для извлекаемой версии;
- задается директория, куда будет выполнено извлечение.

Рекомендуется дублировать извлеченную рабочую копию:
- с основной рабочей копией осуществляется штатная работа;
- дубль — как некий локальный эталон, с которым можно выполнять сравнения без обращения к репозиторию.

ПОРЯДОК РАБОТЫ

Ежедневный рабочий цикл

Обновление рабочей копии

Выполняется с помощью специальной команды для устранения расхождений между рабочей копией и репозиторием. Большое количество расхождений повышает риск возникновения конфликтных изменений, в которых придется разбираться для их разрешения.

Переключение рабочей копии

Выполняется с помощью специальной команды для переключения рабочей копии на другую ревизию (изменение) или версию/ответвление (например, переключение с версии branches/1.0.x на branches/1.1.x или с ревизии 10 на 8 для версии branches/1.0.x).

При переключении полностью обновляется рабочая копия на новое содержимое, которое будет извлечено из репозитория.

Пред переключением выполняется проверка незафиксированных изменений: если изменения есть, то будет выдано предупреждение о конфликте, который необходимо разрешить — зафиксировать, не фиксировать (забыть) или отменить переключение.

Релокация рабочей копии

Аналогично переключению, только изменяется базовый путь к репозиторию.
Релокация возможна на аналогичный по имени репозиторий.

Штатная работа с рабочей копией

Внесение изменение в документы (файлы) проекта.

Фиксация изменений рабочей копии

Завершив очередной этап над заданием, необхоодимо зафиксировать сделанные изменения в репозитории (желательно выполнять: несколько раз в день после значимых изменений и один раз по окончании рабочего дня).

Выполняется с помощью специальной команды, при этом:
- выполняется сравнение версии рабочей копии и версии репозитория;
- если есть конфликт изменений, то его необхомо разрешить;
- выводится информация по выполненному сравнению изменений
  (в виде списка файлов и директорий с пометками: добавлено, изменено, удалено;
  для измененных файлов можно просмотреть измененные фрагменты, которые
  будут зафиксированы);
- пользователь должен задать комментарий (краткое описание) к изменению
  и подтвердить факт фиксации.

Фиксация создает новую ревизию для версии.

ПОРЯДОК РАБОТЫ

Ветвление

Выполняется с помощью специальной команды для создания именованных рабочих версий проекта в репозитории (например, работали с версией из branches/1.0x и на основе нее необходимо сделать значительные изменения; чтобы сохранить версионность — от branches/1.0x делается ответвление с именем branches/2.0.x, с которым будет выполняться далее работа).

Слияние версий

Выполняется с помощью специальной команды для сведения двух версий в одну: исходной версии в целевую (например, слияние рабочей версии branches/2.0.x с главной версией main).

При этом, по сути, выполняется фиксация изменений (ревизий) исходной версии в целевую со всеми сопутствующими действиями: сравнение, разрешение конфликтов, ввод комментария и подтверждение слияния.

Создание релиза

По сути, это аналог ветвления (в разных системах управления версиями выполняетя по разному: например, в subversion выполняется командами ветвления):

- сначала рабочая версия (например, branches/2.0.x) сливается с главной (main);
- далее от главной версии (main) создается ответвление-релиз (tags/2.0.0);
- создается ответвление для следующей рабочей версии (например, branches/3.0.x).

Блокировка / разблокировка

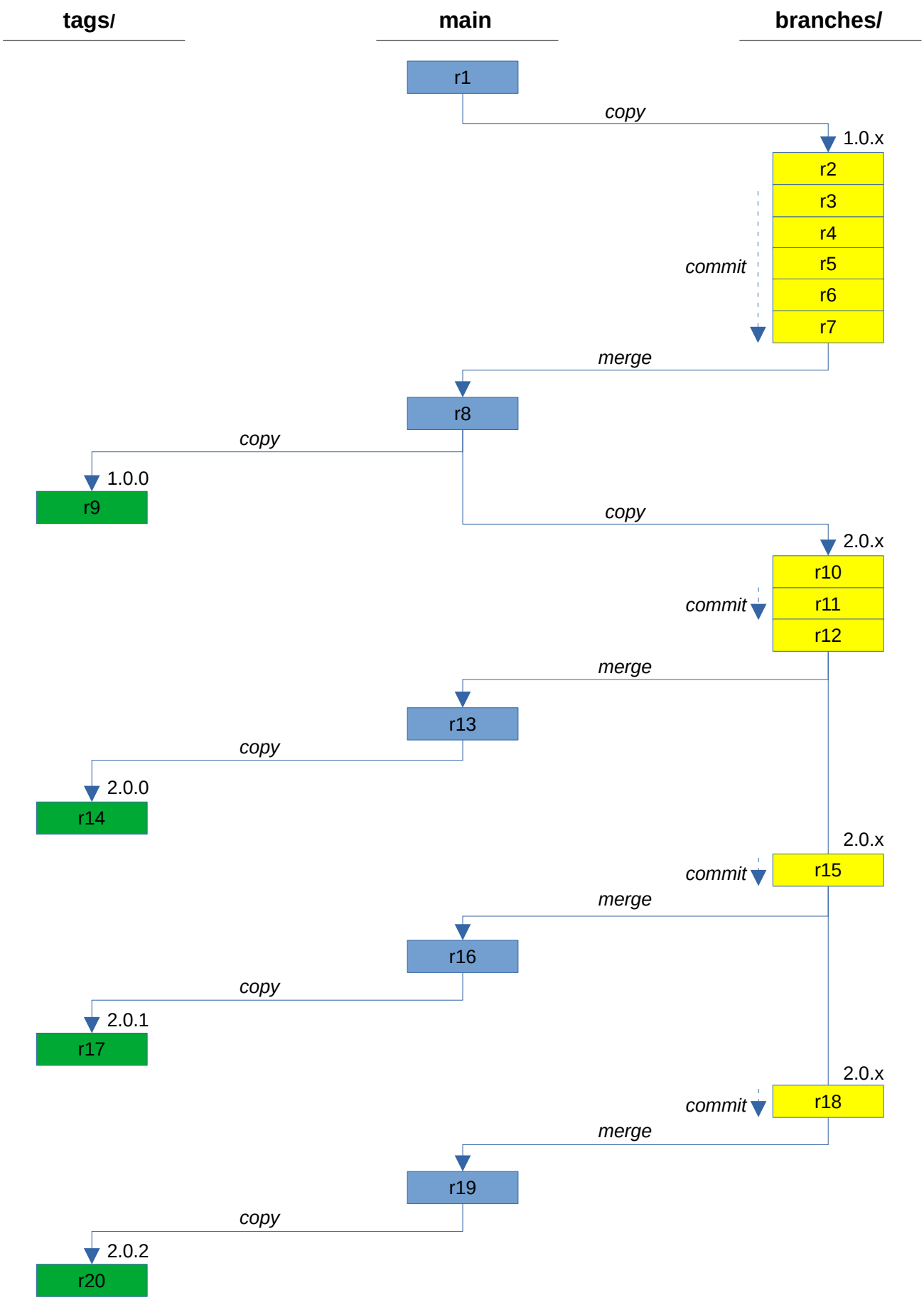
Выполняется с помощью специальной команды и позволяет разработчику захватить в монопольное использование определенный ресурс репозитория (документ или весь проект).

Пользователь, выполнивший блокировку, может работать с ресурсом как обычно (чтение/запись); остальные же пользователи могут работать с ресурсом только на чтение (могут: выполнять ветвления от него, создавать релизы; не могут: фиксировать свои изменения в него, выполнять слияния с ним).

ГРАФ ИЗМЕНЕНИЙ

Пример графа истории изменений проекта

где, rX — номер ревизии (изменения)



GIT

Git

- Открытое и свободно распространяемое программное обеспечение (https://git-scm.com/)
- Распределенная система управления версиями.
- Кроссплатформенная (сборки для различных ОС).
- Реализован на языке Си.
- Только консольный режим (визуализации нет).
- Многоязычный.
- Первый выпуск — 2005 г.

Изначально проект Git был создан Линусом Торвальдсом (программист, создатель и руководитель группы разработки ядра ОС Linux) для разработки ядра Linux.

Проект спроектирован как набор консольных программ (без визуализации).

Но, на данный момент, уже разработано множество графических интерфейсов для данной системы, среди них: TortoiseGit (рассмотрен в следующем разделе).

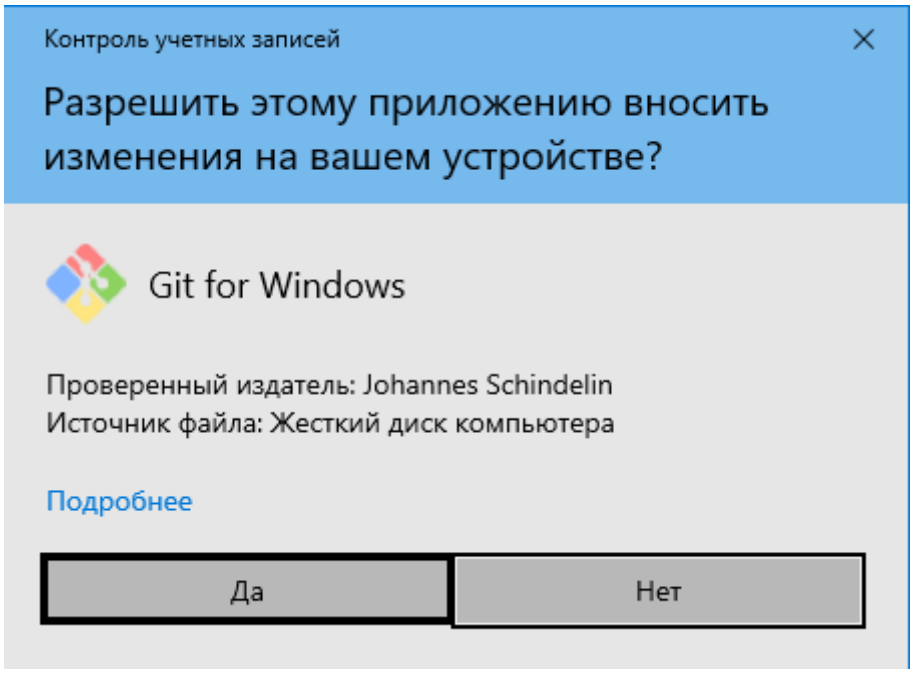
Далее будет рассмотрена установка Git 2.35.1.2-64bit:

- в ОС Windows 10 x64

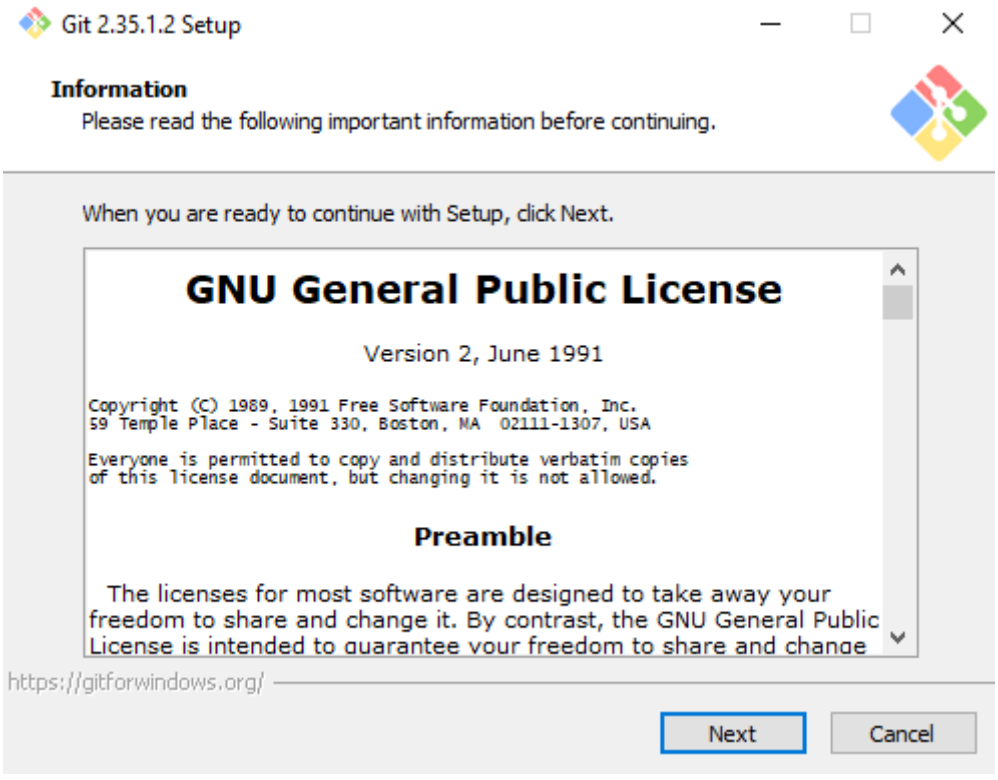
GIT: УСТАНОВКА

1. Запустить установку.

В процессе установки может быть выдано окно системы безопасности, где необходимо дать разрешение на запуск/использование компонента Git, нажав на кнопку «Да».

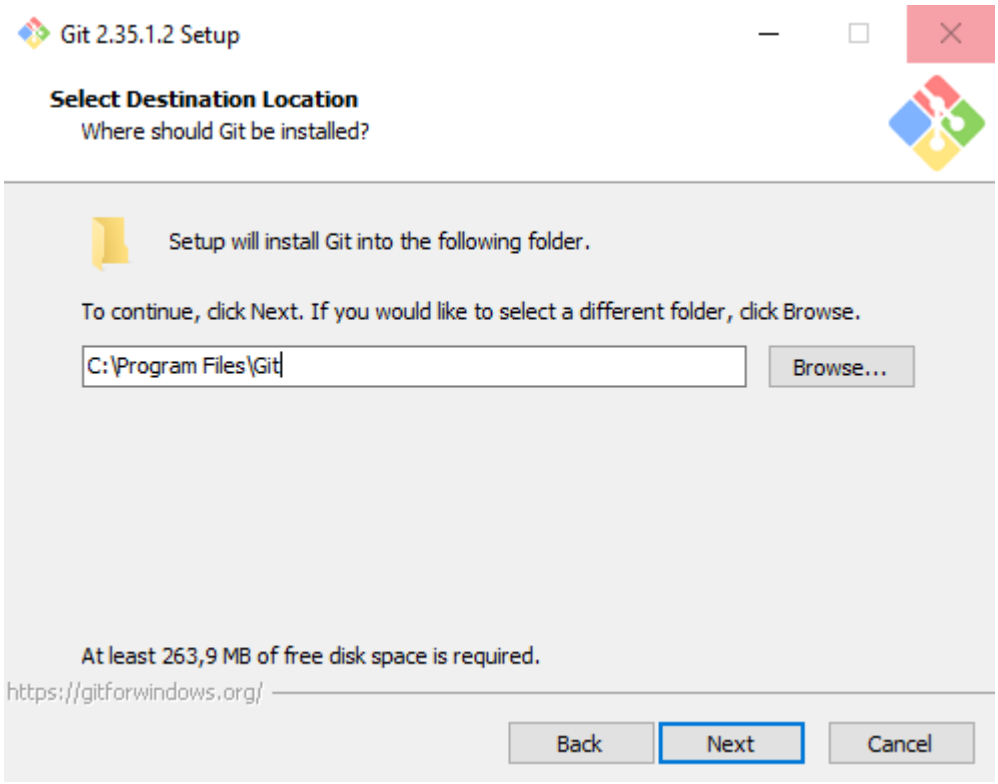


2. В окне «Information»
   a) пролистать текст лицензии,
   b) нажать на кнопку «Next».

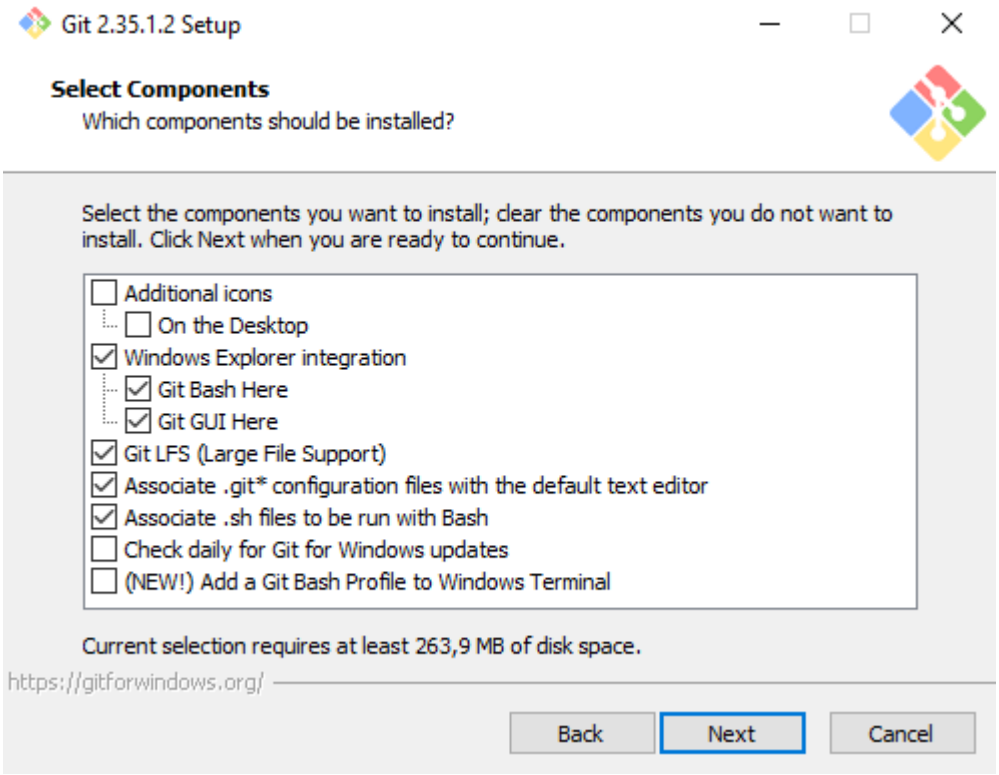


GIT: УСТАНОВКА

3. В окне «Select Destination Location»
   a) задать путь установки,
   b) нажать на кнопку «Next».

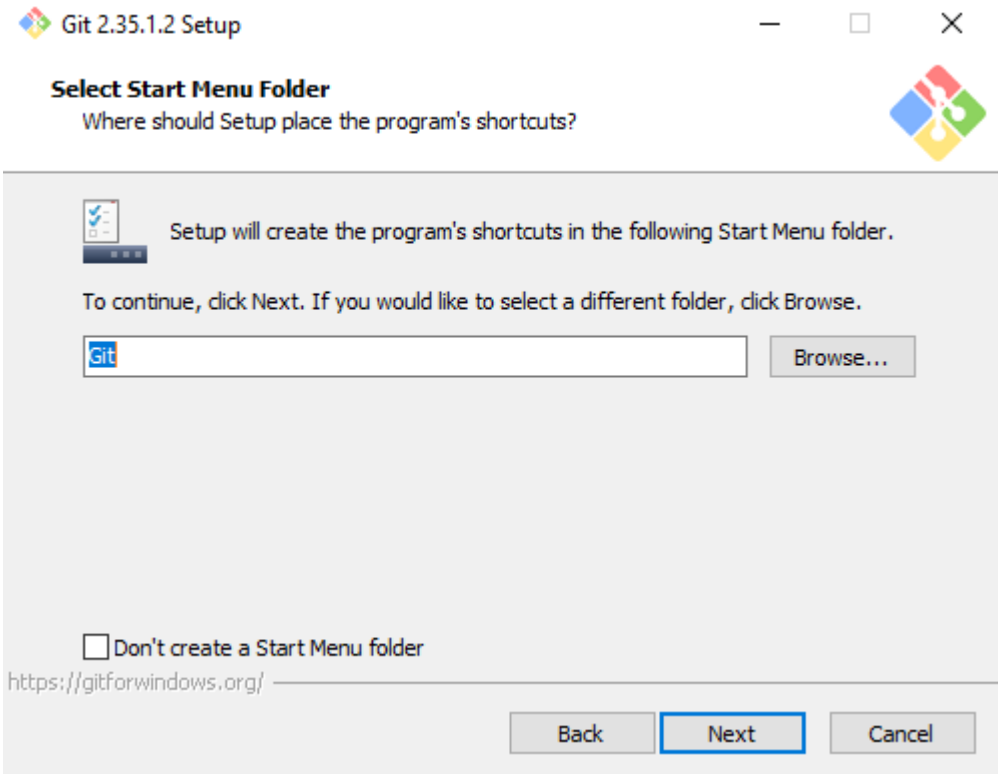


4. В окне «Select Components»
   a) выбрать необходимые настройки (можно оставить как есть),
   b) нажать на кнопку «Next».

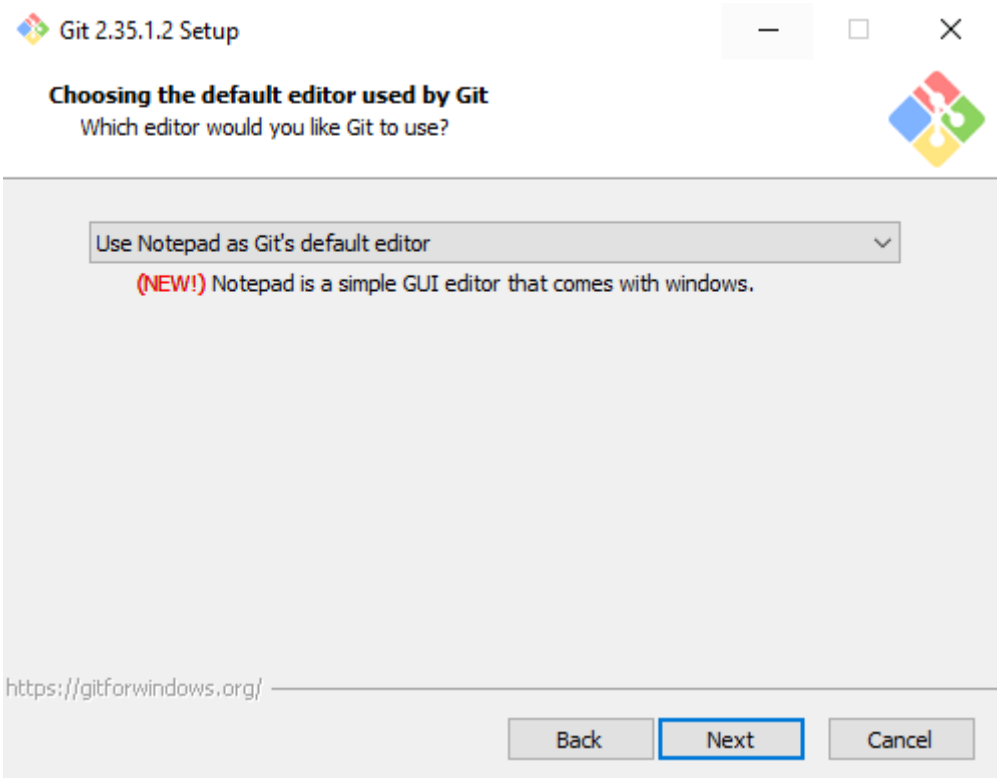


GIT: УСТАНОВКА

5. В окне «Select Start Menu Folder»
   a) задать настройки интеграции Git в Стартовое меню ОС Windows,
   b) нажать на кнопку «Next».

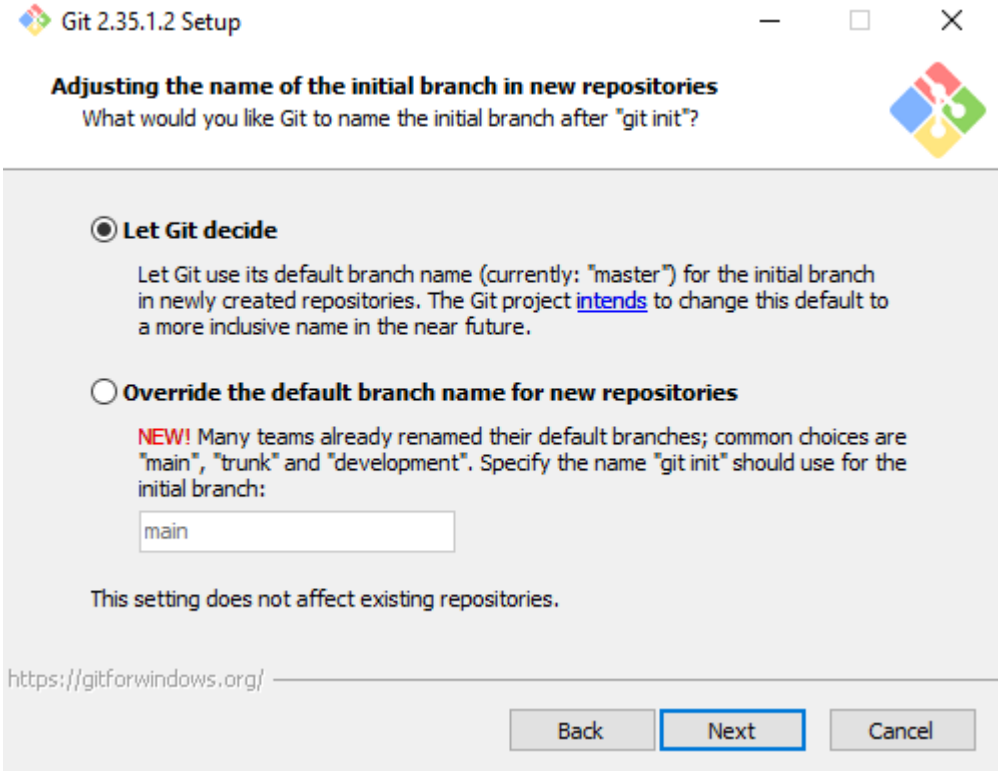


6. В окне «Choosing the default editor used by Git»
   a) выбрать текстовый редактор (NotePad), который будет привязан к Git (например, для ввода комментариев при фиксации изменений и т.д.)
   b) нажать на кнопку «Next».



GIT: УСТАНОВКА

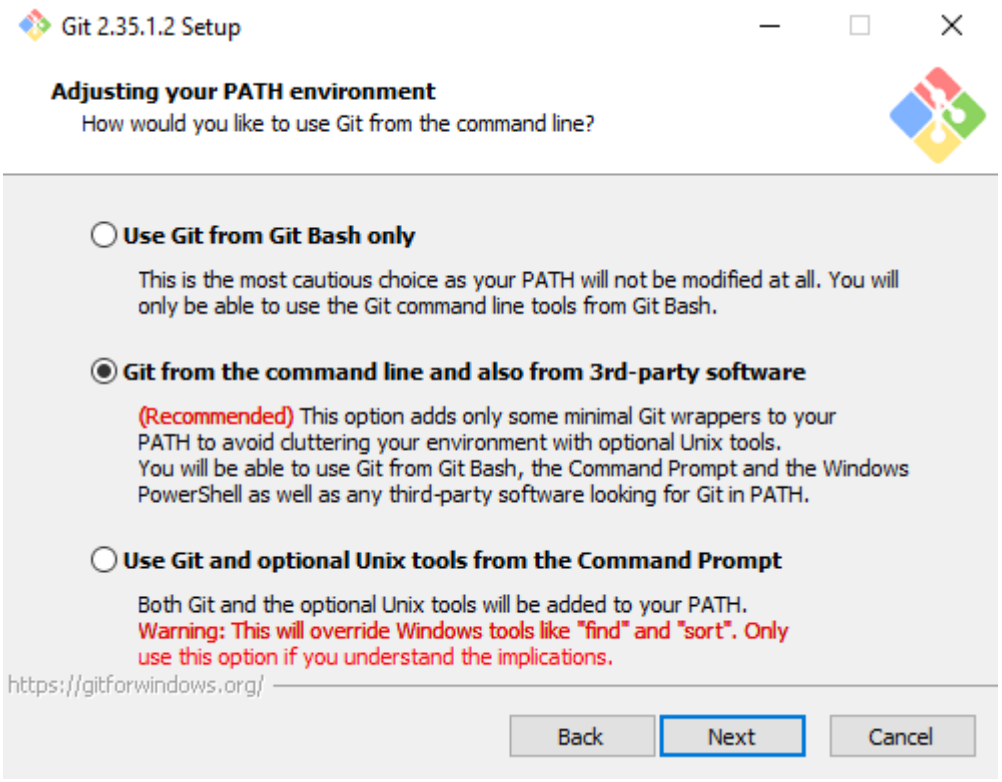
7. В окне «Adjusting the name of the initial branch in new repositories»
   a) указать имя по-умолчанию для ствола нового репозитория
      (Let Git decide - имя «master» по стандарту Git)
   b) нажать на кнопку «Next».



GIT: УСТАНОВКА

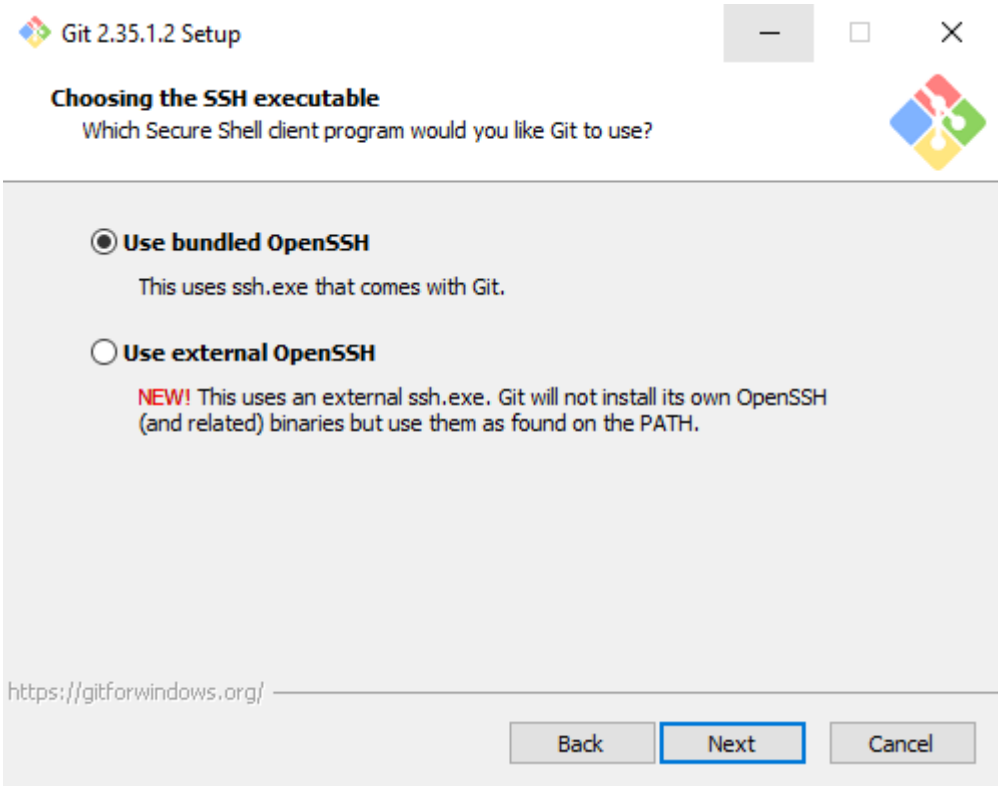
8. В окне «Adjusting your PATH environment»
   a) указать, что Git будет исполняться в консольном режиме, а также вызываться сторонними приложениями (Git from the command line and also ...)
   b) нажать на кнопку «Next».

В переменную окружения PATH будет добавлен путь к исполняемому файлу git.exe — это необходимо для работы клиента TortoiseGit.

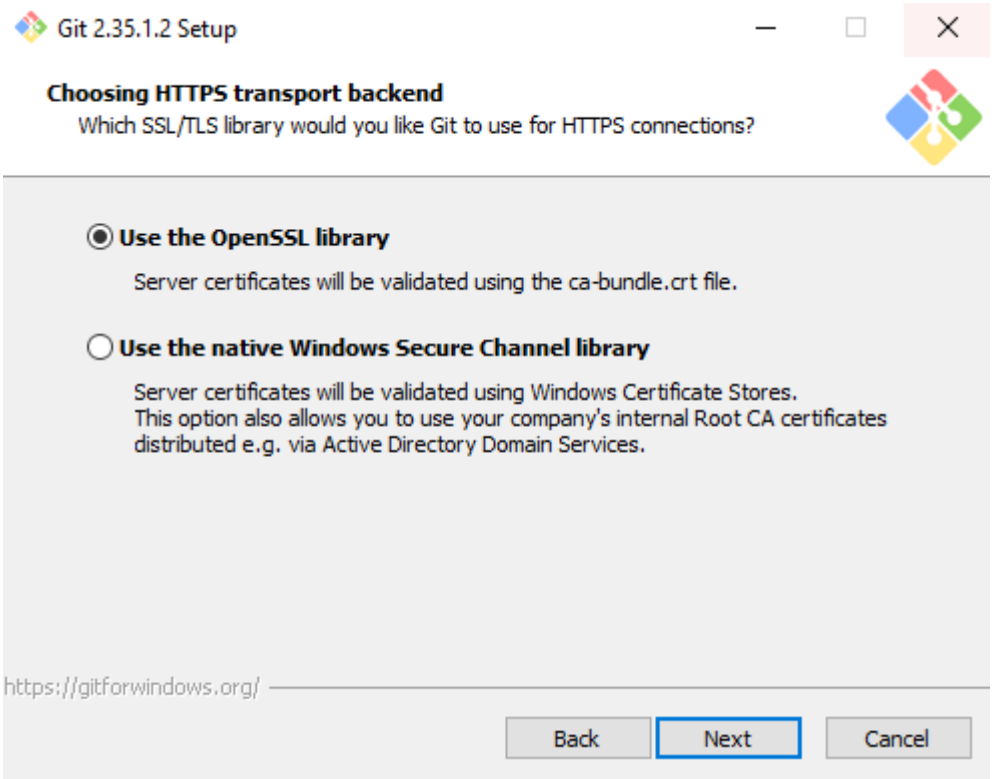


GIT: УСТАНОВКА

9. В окне «Choosing the SSH executable»
   a) выбрать, какой SSH использовать
      (Use bundled OpenSSH — поставляемый с Git)
   b) нажать на кнопку «Next».

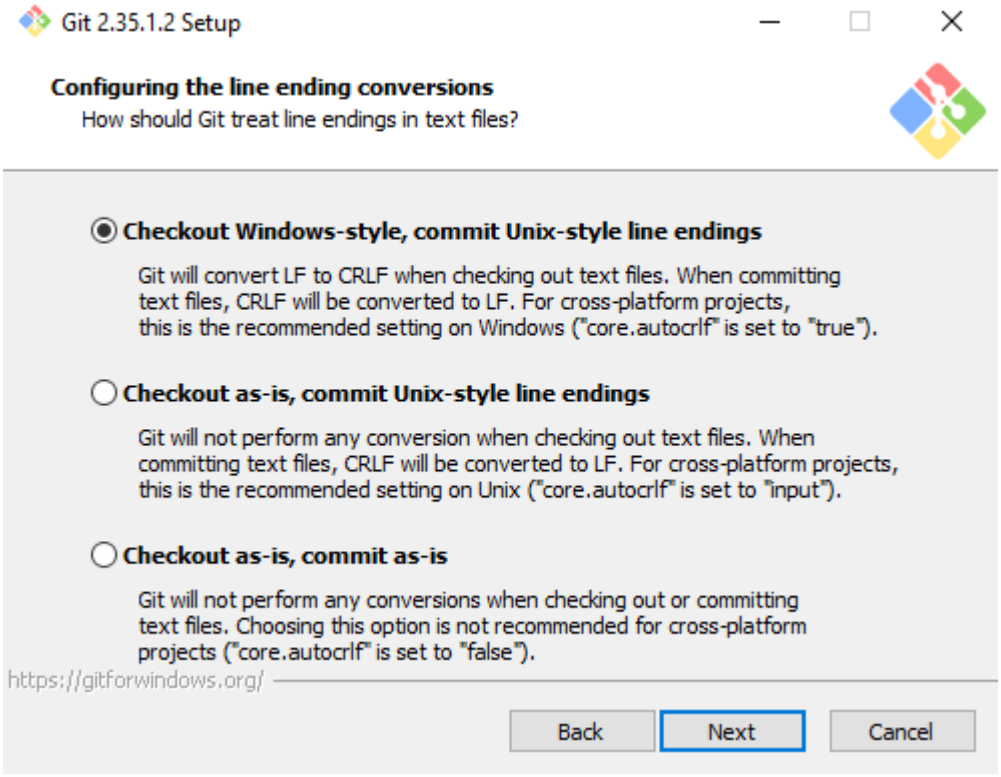


10. В окне «Choosing HTTPS transport backend»
    a) оставить все как есть,
    b) нажать на кнопку «Next».

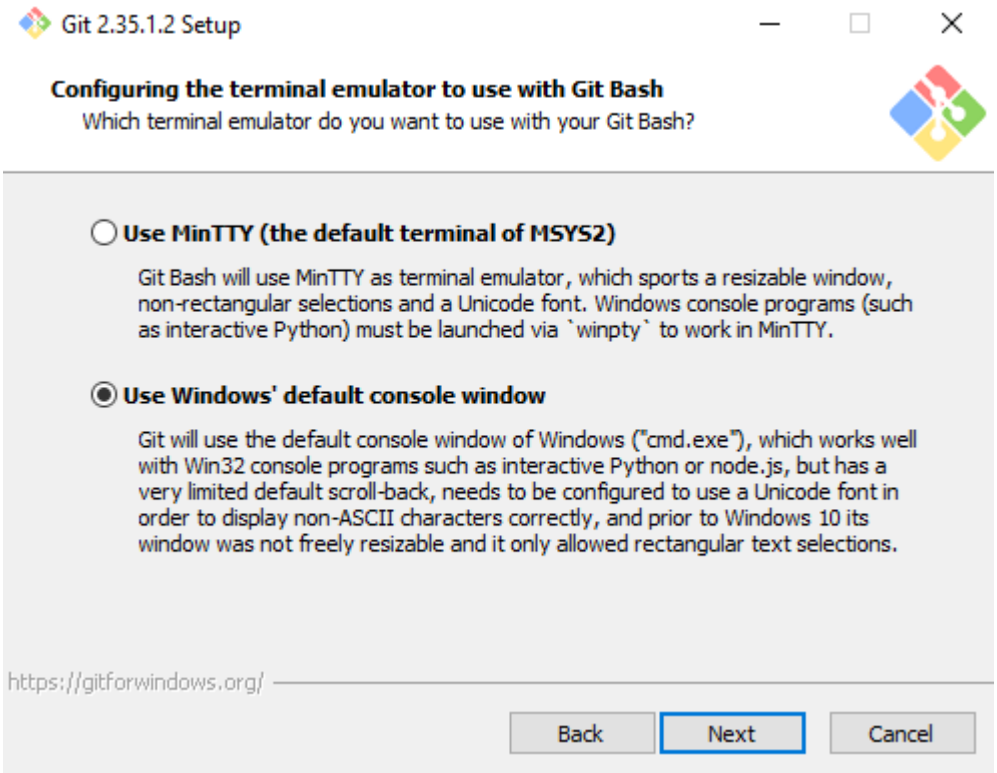


GIT: УСТАНОВКА

11. В окне «Configuring the line ending conversions»
    a) выбрать, какие будут использованы симолы перевода строки в комментариях (оставить как есть)
    b) нажать на кнопку «Next».

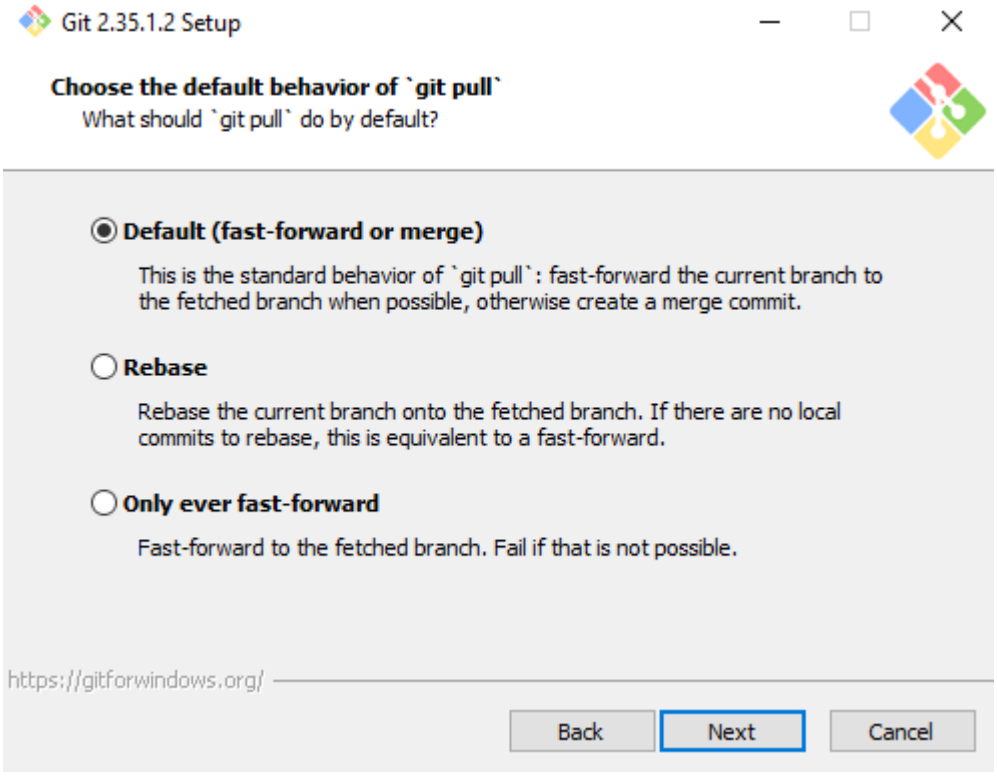


12. В окне «Configuring the terminal emulator to use with Git Bash»
    a) выбрать, какую программу-терминал будет использовать Git (Use Windows' default...)
    b) нажать на кнопку «Next».

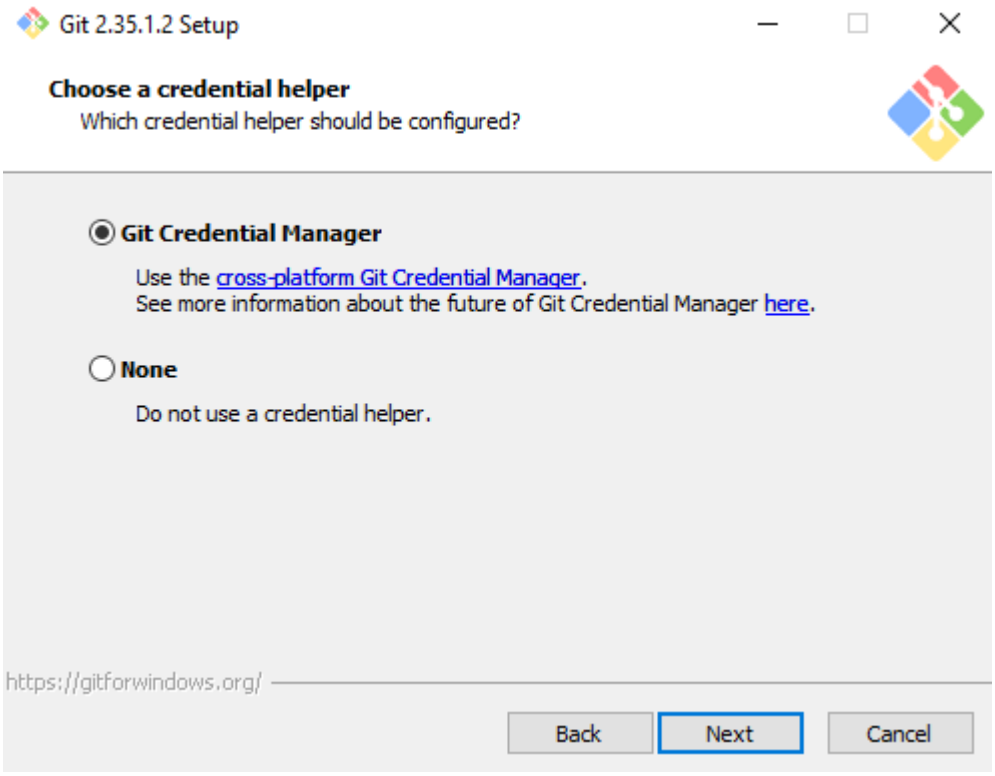


GIT: УСТАНОВКА

13. В окне «Choosing the default behavior of `git pull`»
    a) задать, как должна работать команда `git pull` (оставить как есть)
    b) нажать на кнопку «Next».

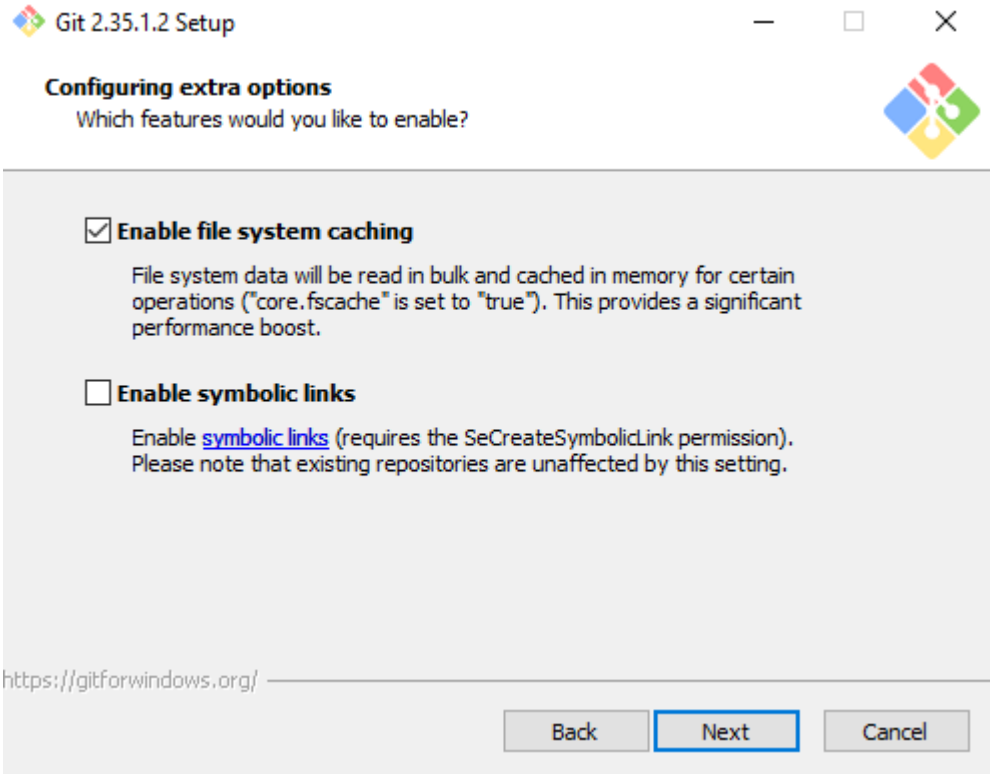


14. В окне «Choose a credential helper»
    a) указать, какую программу-помощьник будет использовать Git для управления учетными данными (оставить как есть)
    b) нажать на кнопку «Next».

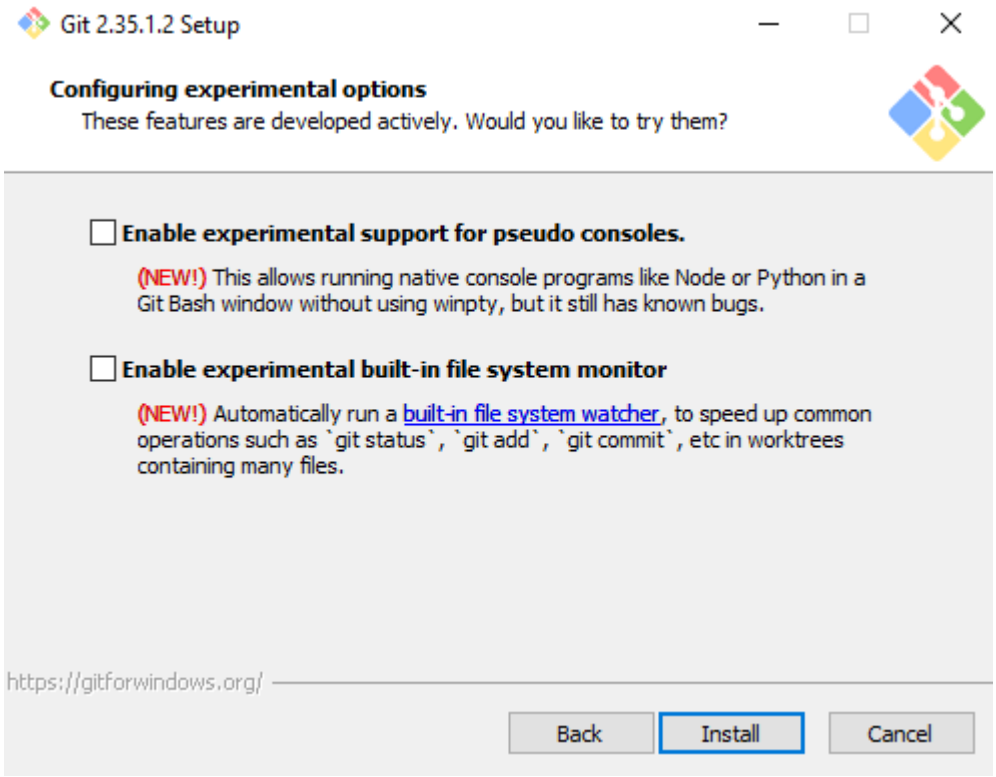


GIT: УСТАНОВКА

15. В окне «Configuring extra options»
    a) включить поддержку кэширования файловой системы (Enable file system caching)
    b) нажать на кнопку «Next».



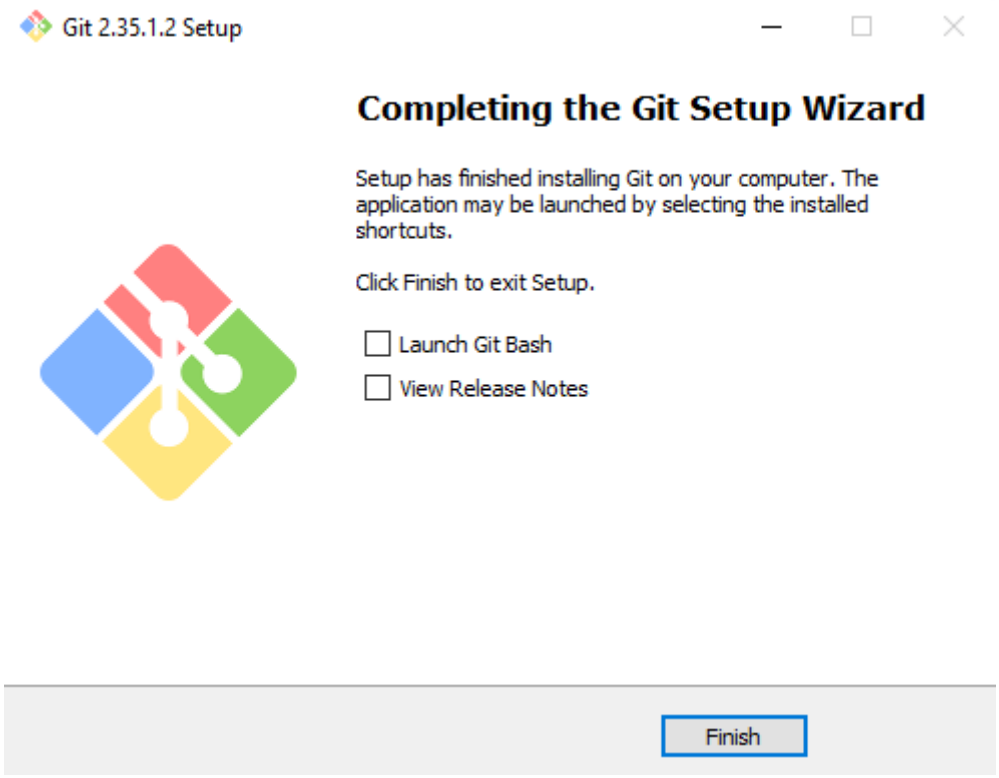
16. В окне «Configuring experimental options»
    a) не выбирать экспериментальные опции,
    b) нажать на кнопку «Install».



GIT: УСТАНОВКА

Дождаться завершения установки.

17. В окне «Completing ...»
    a) не задавать никакие опции,
    b) нажать на кнопку «Finish».



TORTOISE GIT

TortoiseGit

- Открытое и свободно распространяемое программное обеспечение (https://tortoisegit.org/)
- Визуальный клиент для системы управления версиями Git (на основе клиента TortoiseSVN для системы SVN)
- Для ОС Windows.
- Реализован на языке Си++.
- Многоязычный.
- Первый выпуск — 2008 г.
- ***Требуется предварительная установка консольной утилиты git*** *(в комплект не входит)*

Реализован как расширение проводника Windows:

- Встроен в контекстное меню (вызывается нажатием правой кнопки в директории);
- Добавляет иконки к документам, находящимся под управлением системы, для отображения их статуса (зафиксировано, изменено и незафиксировано и т. п.).

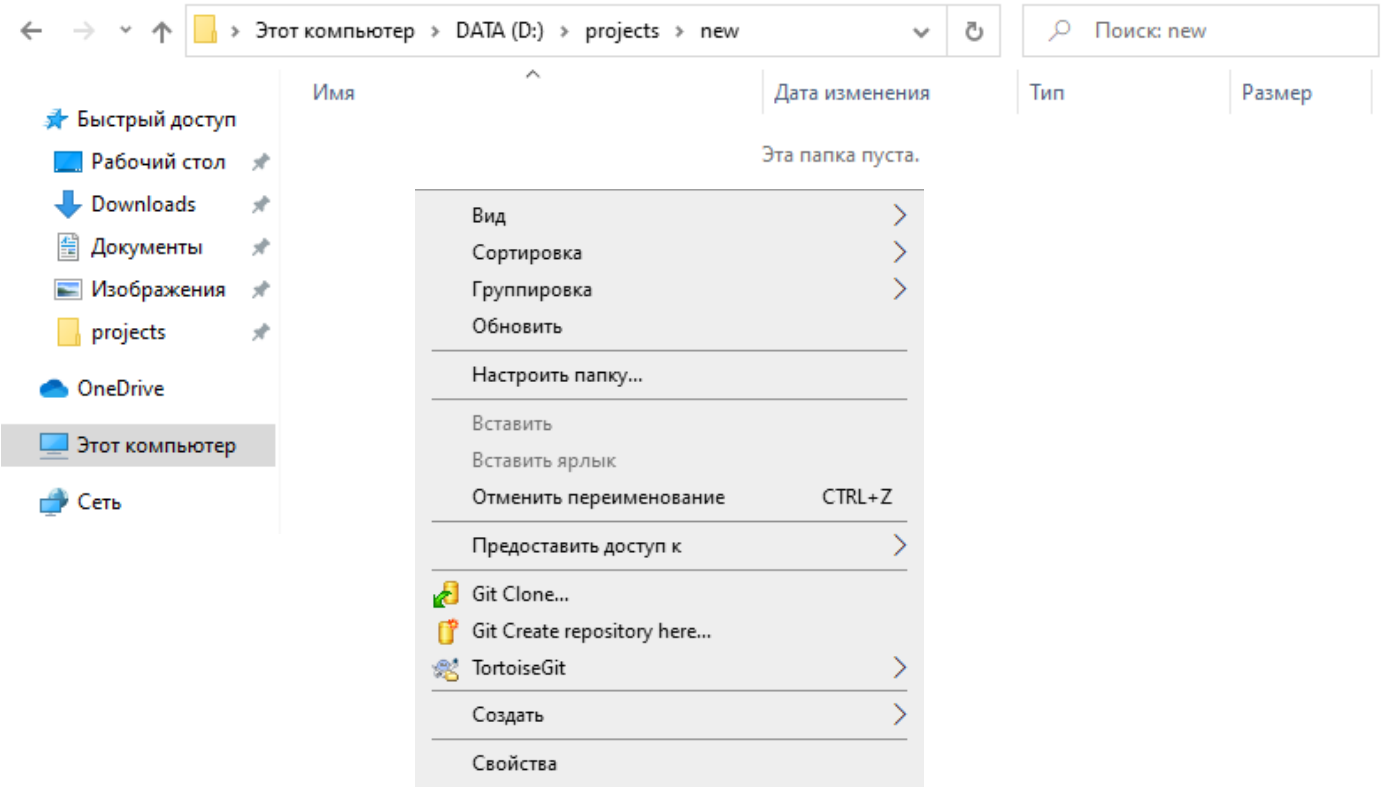
Поддерживает выполнение обычных команд:

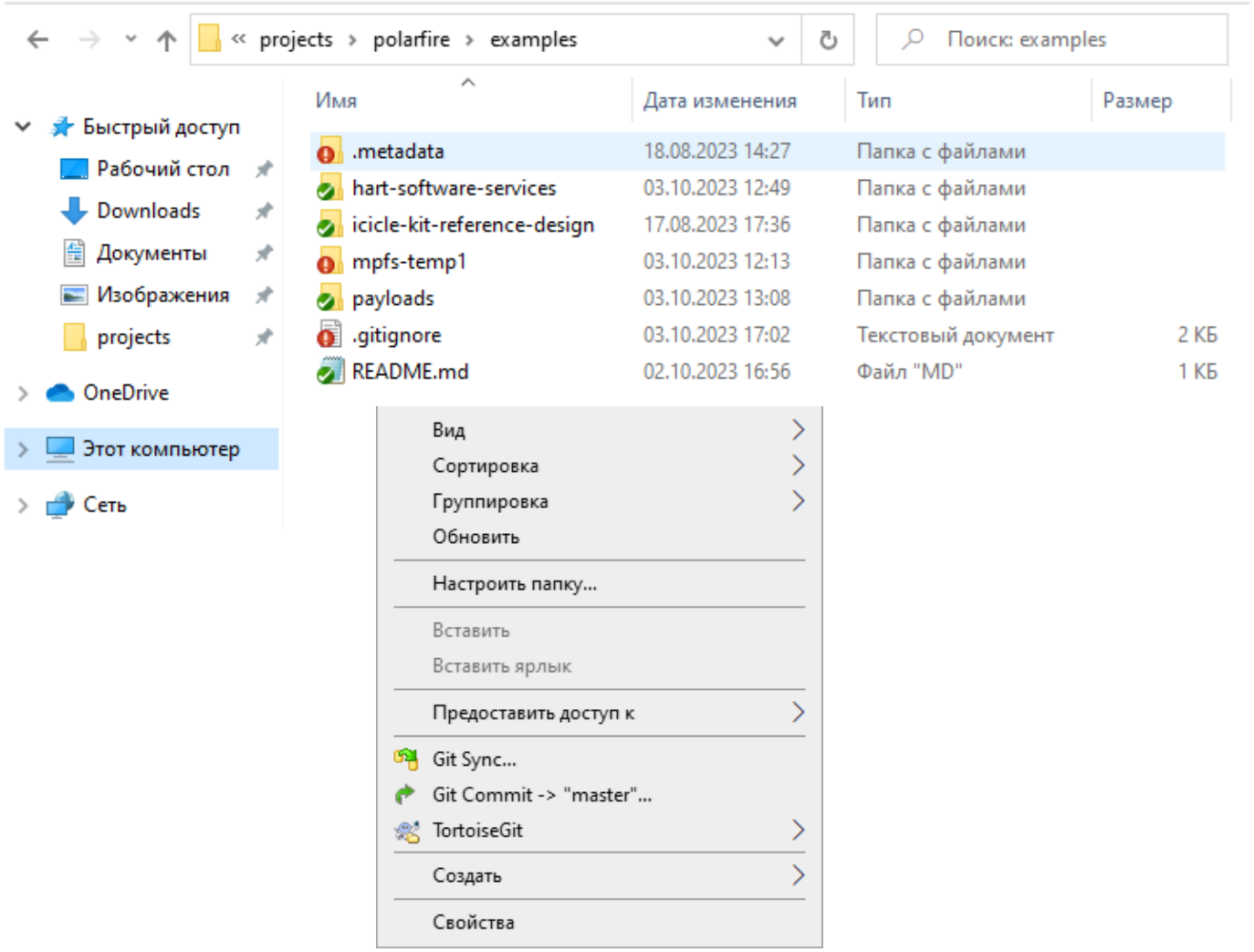
- создание репозитория,
- извлечение (клонирование) рабочей копии,
- релокация рабочей копии,
- просмотр журнала истории изменений,
- просмотр графа изменений,
- сравнение двух версий,
- фиксация изменений,
- разрешение конфликтов,
- ветвление,
- слияние,
- блокирование,
- синхронизация с основным репозиторием,
- настройки,
- и пр.

Далее будет рассмотрена установка и работа с TortoiseGit 2.13.1-64bit:

- в ОС Windows 10 x64
- В простом локальном режиме:
- без применения средств SSH (приватных и публичных ключей, шифрофанного канала передачи данных и т. п.)
- локальный репозиторий
- однопользовательский режим

TORTOISE GIT



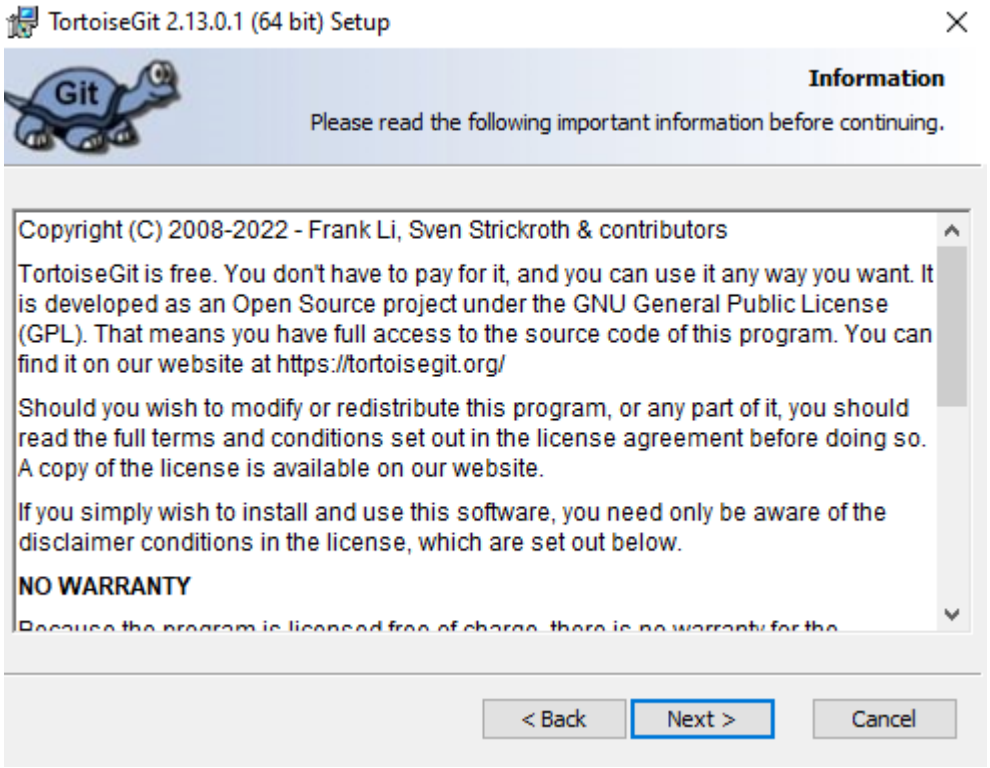


TORTOISE GIT: УСТАНОВКА

1. Запустить установку.

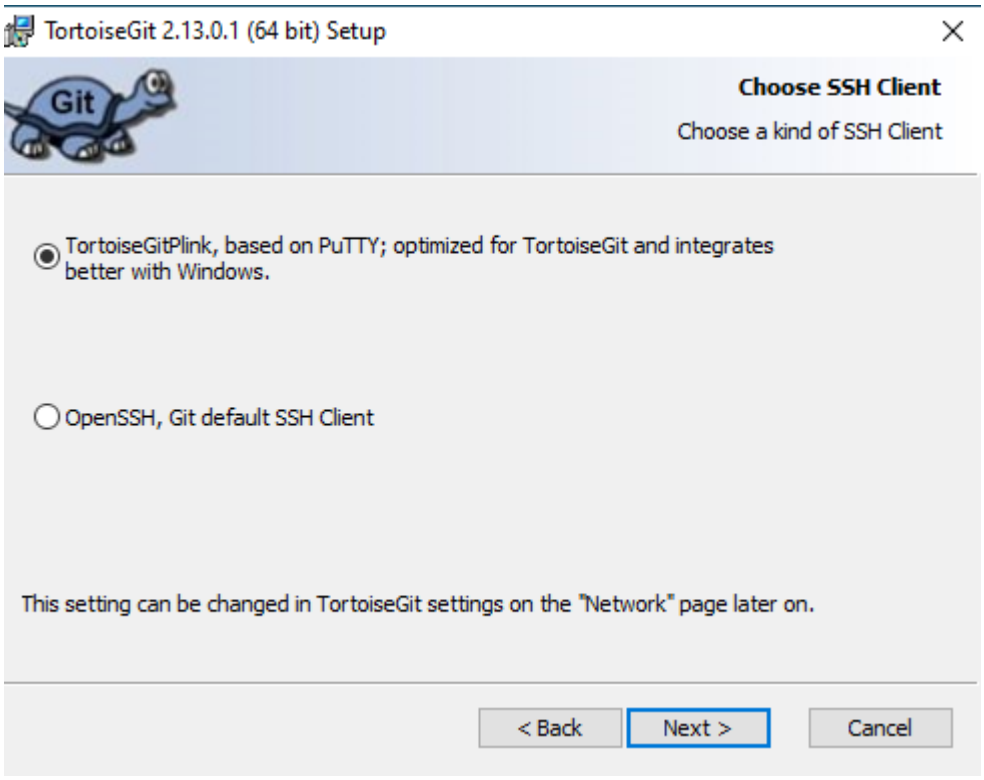
2. В окне «Welcome to the TortoiseGit ...»:
   a) нажать на кнопку «Next».

3. В окне «Information»:
   a) пролистать описание,
   b) нажать на кнопку «Next».

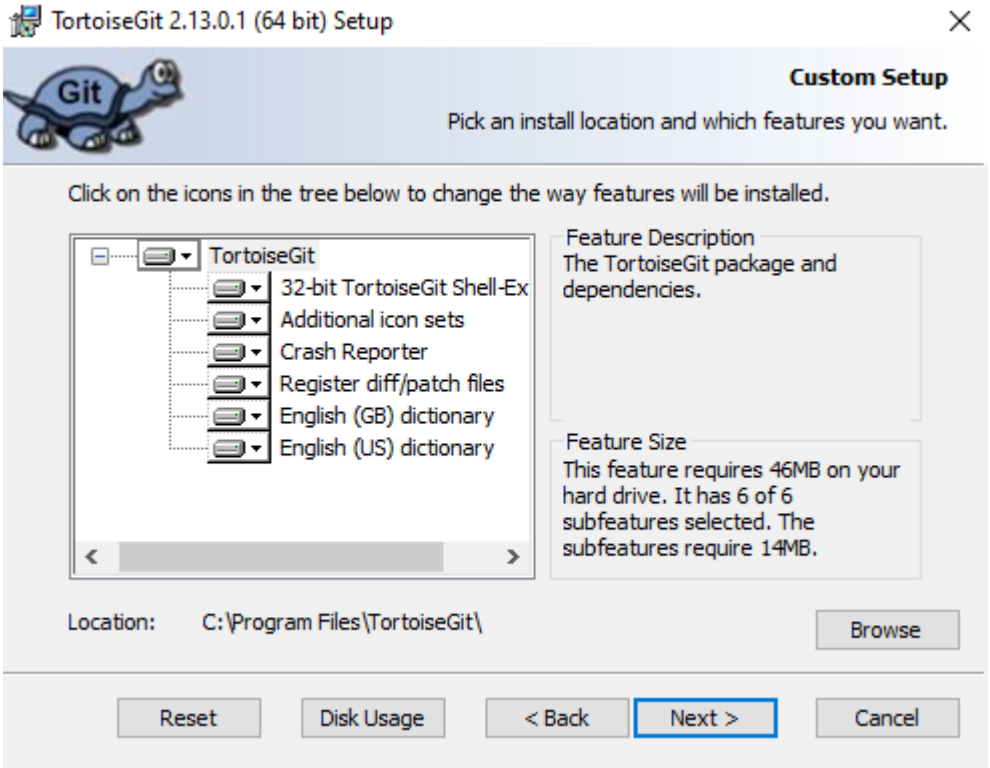


TORTOISE GIT: УСТАНОВКА

4. В окне «Choose SSH Client»
   a) выбрать пункт «TortoiseGitPlink ...» (отмечен по-умолчанию),
   b) нажать на кнопку «Next».

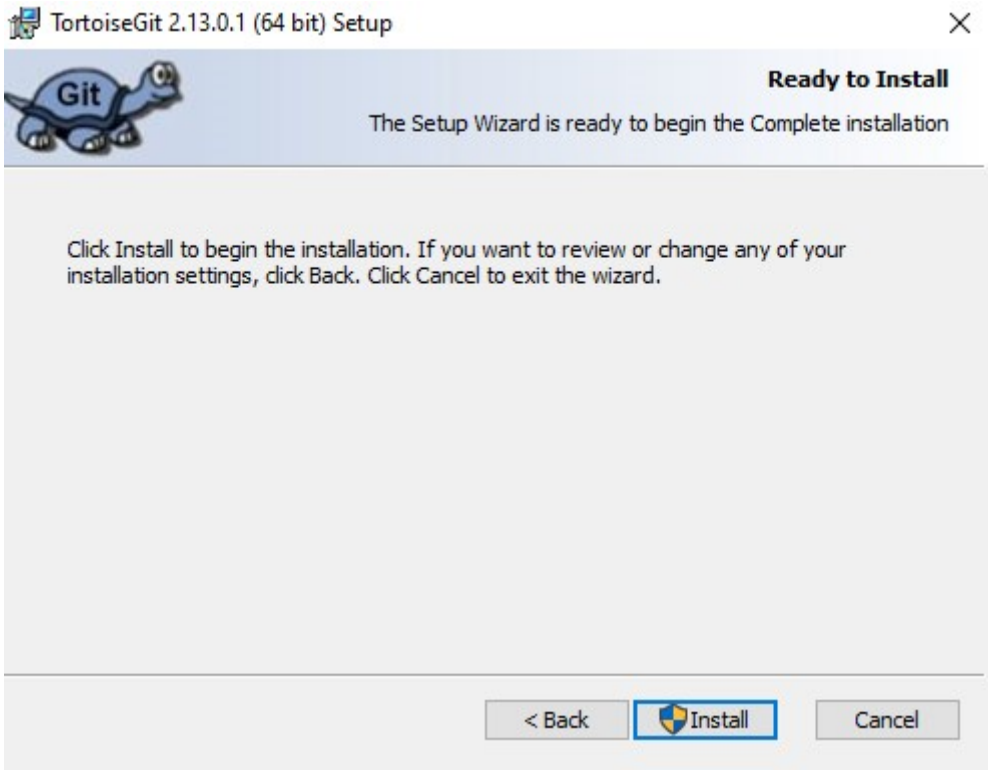


5. В окне «Custom Setup»
   a) задать путь установки (Location),
   b) нажать на кнопку «Next».



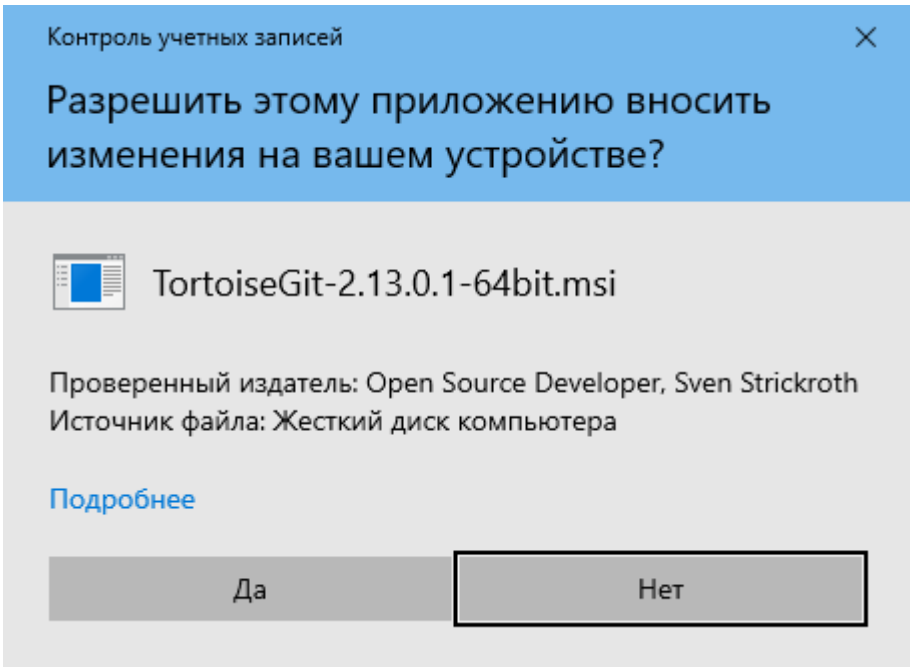
TORTOISE GIT: УСТАНОВКА

6. В окне «Ready to Install»
   а) нажать на кнопку «Install».



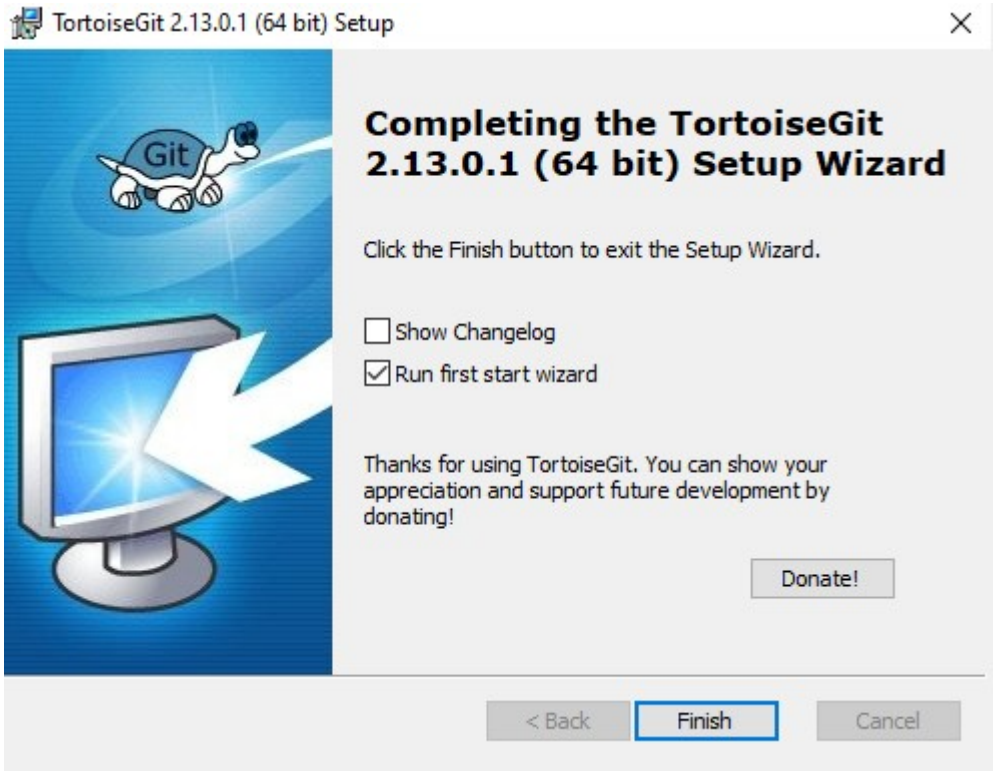
Дождаться завершения установки.

В процессе установки может быть выдано окно системы безопасности, где необходимо дать разрешение на запуск/использование компонента TortoiseGit, нажав на кнопку «Да».



TORTOISE GIT: УСТАНОВКА

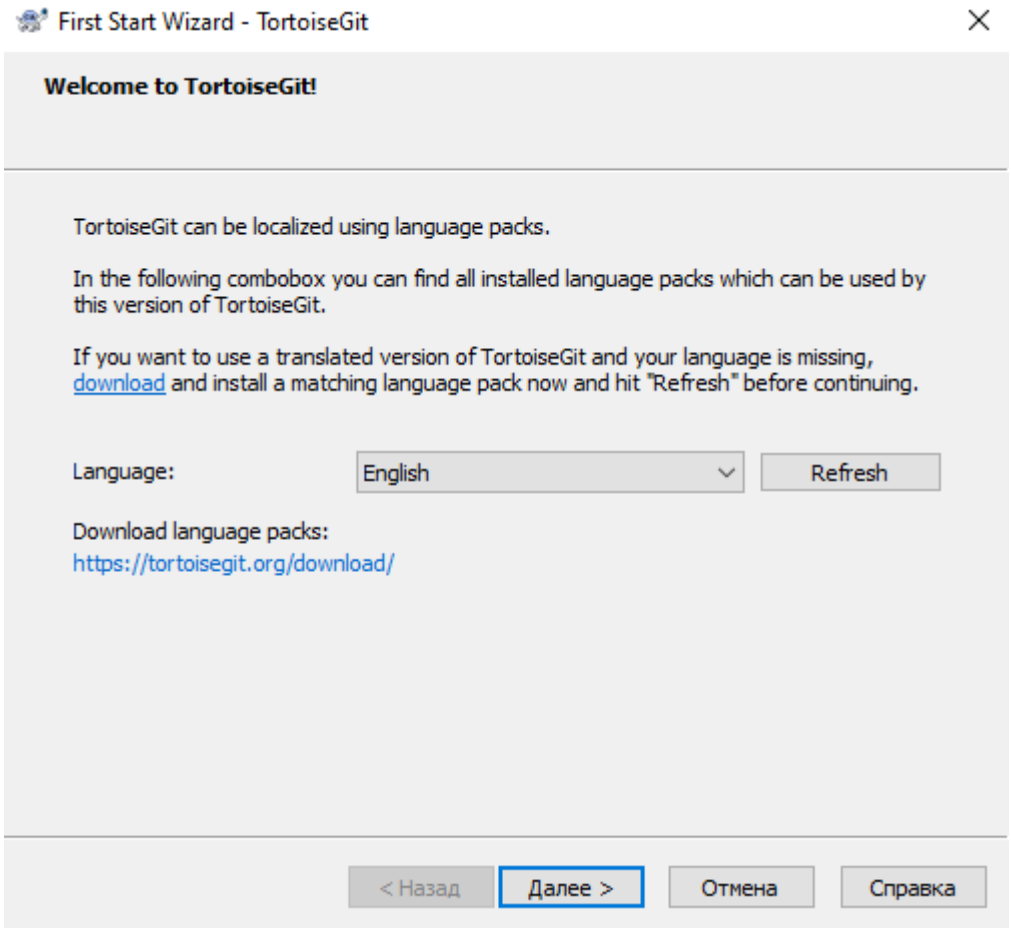
8. В окне «Completing the TortoiseGit ...»
   a) установить отметку «Run first start wizard»,
   b) нажать на кнопку «Finish».



TORTOISE GIT: УСТАНОВКА

9. В окне «Welcome to TortoiseGit!»
   a) выбрать язык «English»,
   b) нажать на кнопку «Далее».

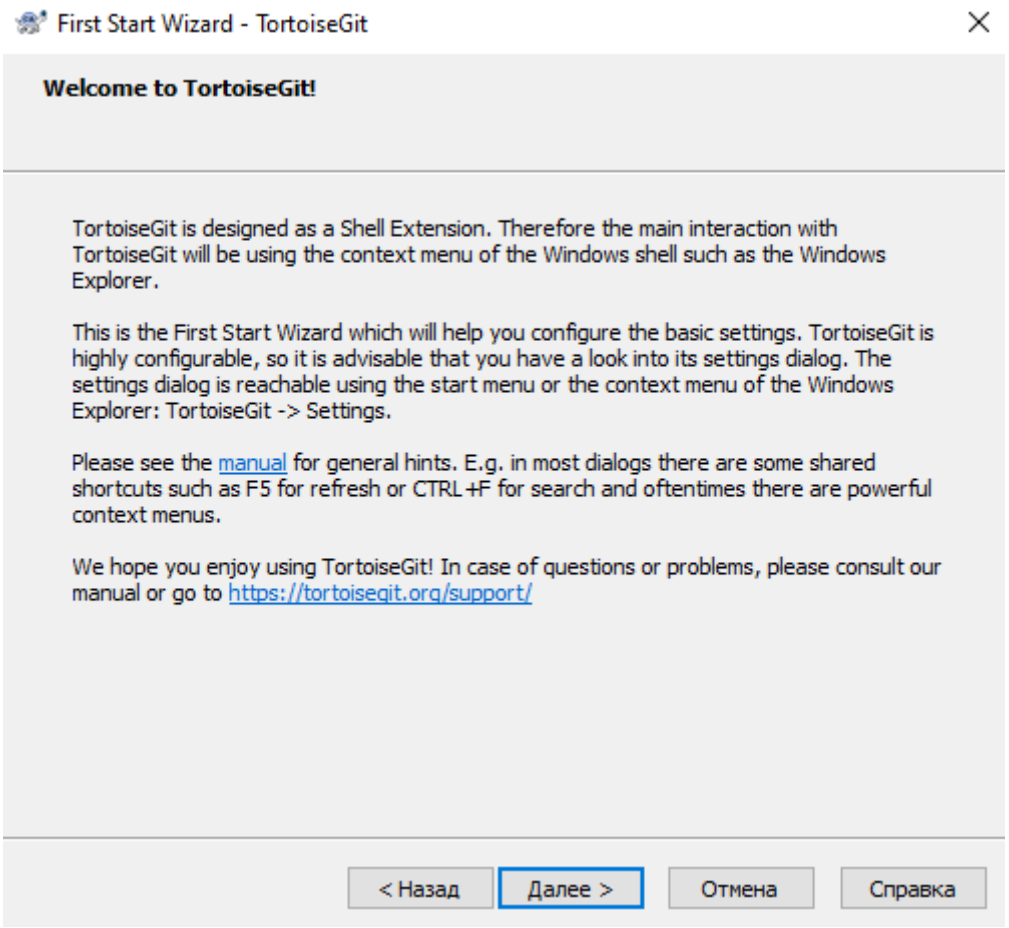
По-умолчанию, в списке языков нет Русского — его можно установить отдельно (ссылки на скачивания языковых пакетов приведены в данном окне)



TORTOISE GIT: УСТАНОВКА

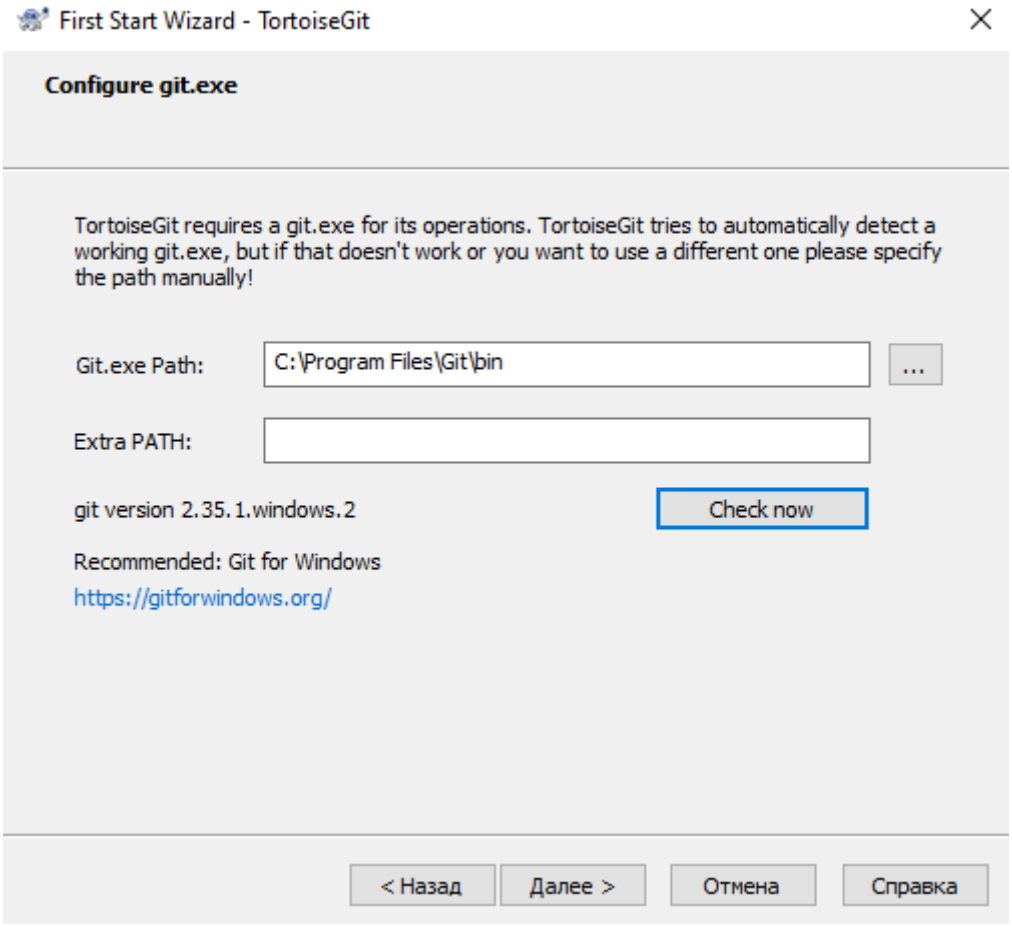
10. В окне «Welcome to TortoiseGit!» дано описание, что:
    a) TortoiseGit является расширением Проводника ОС Windows (контекстное меню),
    b) настройки клиента в меню «Settings»,
    c) прочитать инструкцию можно, перейдя по ссылке, указанной в данном окне,
    d) и т. д.

Для продолжения — нажать на кнопку «Далее».



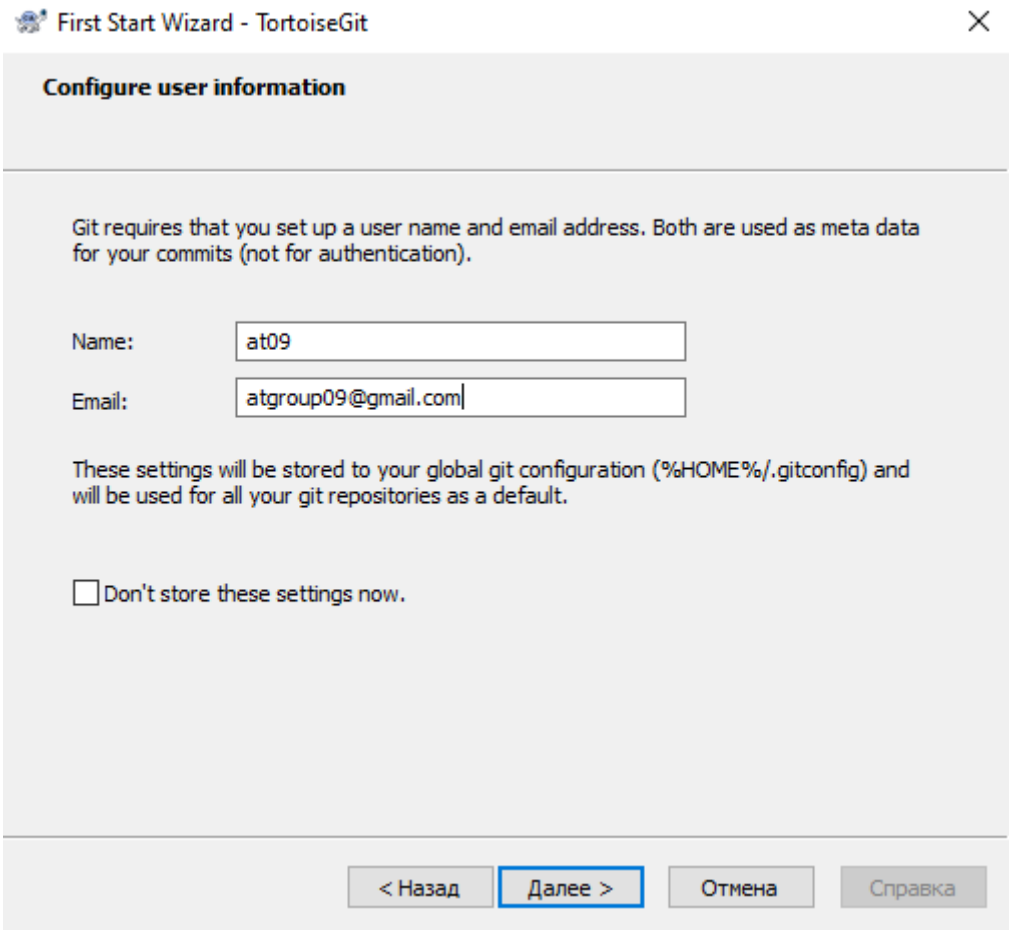
TORTOISE GIT: УСТАНОВКА

11. В окне «Configure git.exe»:
    a) указать путь, где находится исполняемый файл консольной утилиты git.exe (если Git ранее был установлен, то путь будет автоматически заполнен)
    b) для проверки нажать на кнопку «Check now»,
    c) нажать на кнопку «Далее».



TORTOISE GIT: УСТАНОВКА

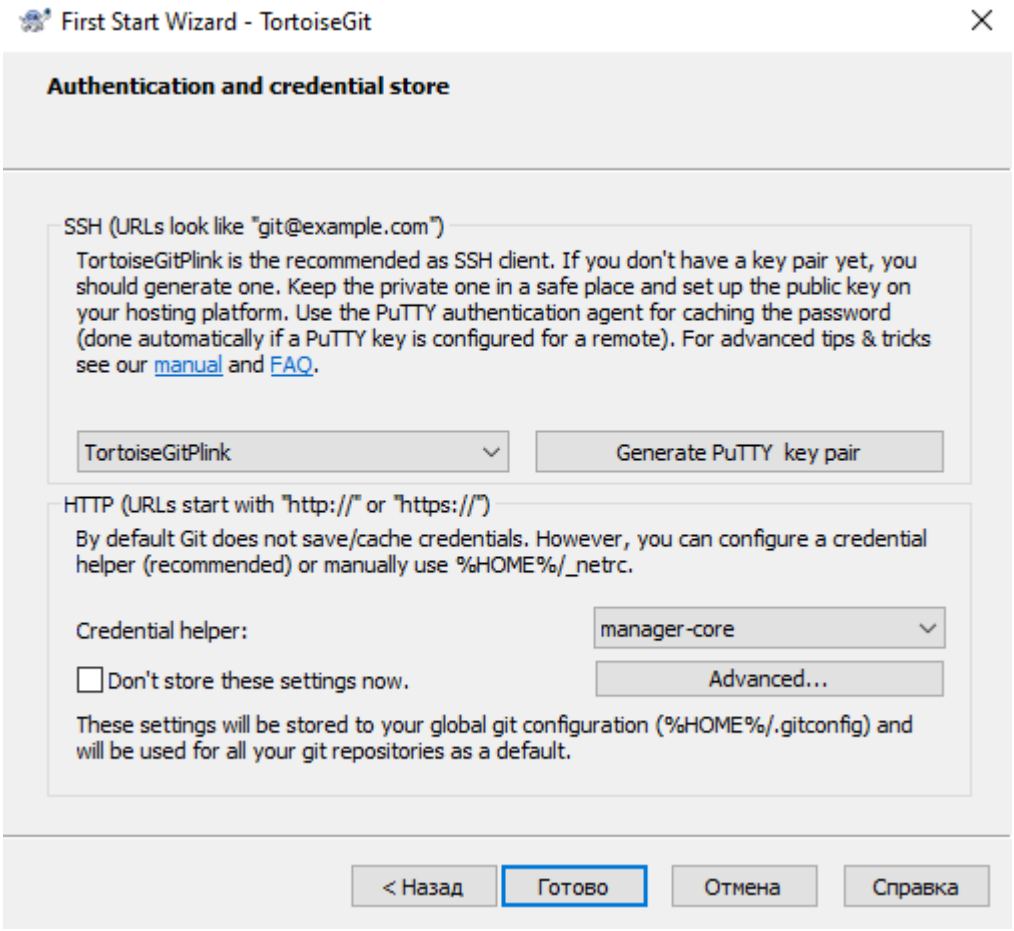
12. В окне «Configure user information»:
    a) ввести свои Имя (псевдоним) и e-mail
       (они будут предгататься по-умолчанию при фиксации изменений)
    b) нажать на кнопку «Далее».



TORTOISE GIT: УСТАНОВКА

13. В окне «Authentication and credential store»:
    a) ничего не менять,
    b) нажать на кнопку «Готово».

Любые настройки, при необходимости, можно поменять в процессе работы с программой.



TORTOISE GIT: СОЗДАНИЕ РЕПОЗИТОРИЯ

Git предоставляет следующие способы создания репозитория

Репозиторий — рабочая копия

Имеется непустая директория (наполенена файлами) с рабочим проектом.

В директории проекта запускается процесс создания репозитория:
- указывается, что это будет рабочая копия (опция «Make it Bare» выключена),
- автоматически создается поддиректория «.git» (признак, что это рабочая копия).

Далее с директорией проекта можно начинать работать как с рабочей копией:
- вносить изменения в проект (например, изменять содержимое документов)
- фиксировать изменения (коммитить)
- и все остальные действия, предоставленные TortoiseGit
  (кроме синхронизации с основным репозиторием «Git sync» - его нет, в данном случае)

Этот способ является не безопасным (не рекомендуется):
случайно удалив рабочую копию:
- теряется вся история изменений
- рабочая копия не восстанавливается

Репозиторий — хранилище (Bare)

Имеется непустая директория (наполенена файлами) с рабочим проектом.

Для хранилища требуется пустая директория (вне директории проекта).

В пустой директории хранилища запускается процесс создания репозитория:
- указывается, что это будет хранилище (опция «Make it Bare» включена),
- автоматически создается файловая структура пустого хранилища.

Далее извлекается рабочая копия (по-умолчанию пустая, т. к. репозиторий пустой) и в эту пустую директорию копируется рабочий проект.

Далее с рабочей копией можно работать в шатном режиме:
- вносить изменения в проект (например, изменять содержимое документов)
- фиксировать изменения (коммитить)
- и все остальные действия, предоставленные TortoiseGit
  (включая синхронизацию с основным репозиторием «Git sync» - с хранилищем).

Этот способ является типичным и безопасным:
случайно удалив рабочую копию:
- теряется только рабочая копия
- репозиторий с историей изменений остается
- рабочая копия восстанавливается (клонируется) из репозитория
- репозиторий можно перенести на другой ПК (сервер) с доступом к сети
(для рабочей копии, в данном случае, потребуется выполнить релокацию)

TORTOISE GIT: СОЗДАНИЕ РЕПОЗИТОРИЯ

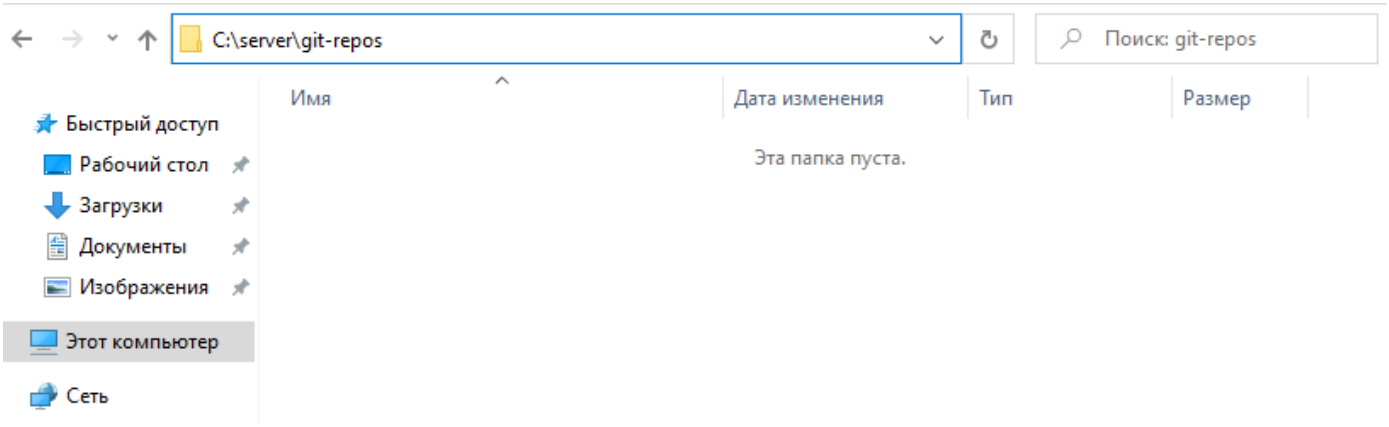
Пример

Задача:

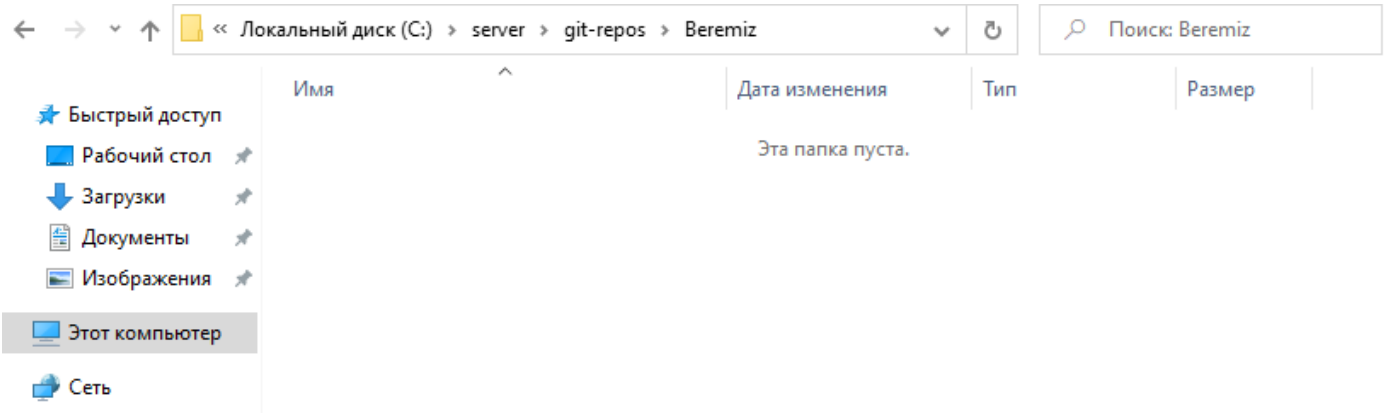
- Имеется проект «Beremiz» (с исходным кодом) в директории `C:\projects\Beremiz`

- Создать репозиторий для этого проекта

1. Создать пустую директорию, где будут размещаться любые репозитории `C:\server\git-repos`

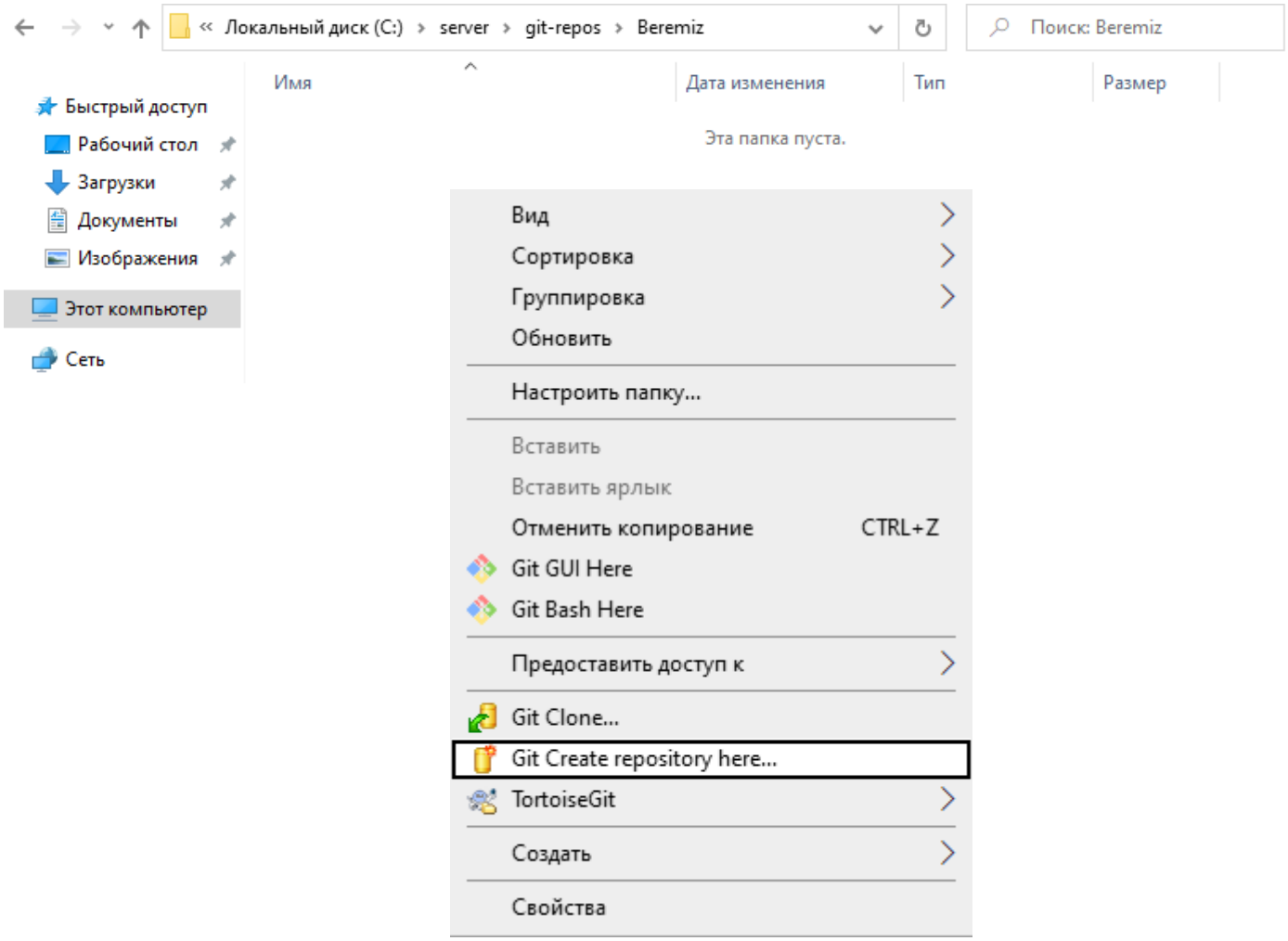


2. Создать пустую директорию для хранилища (репозитория) проекта «Beremiz» `C:\server\git-repos\Beremiz`

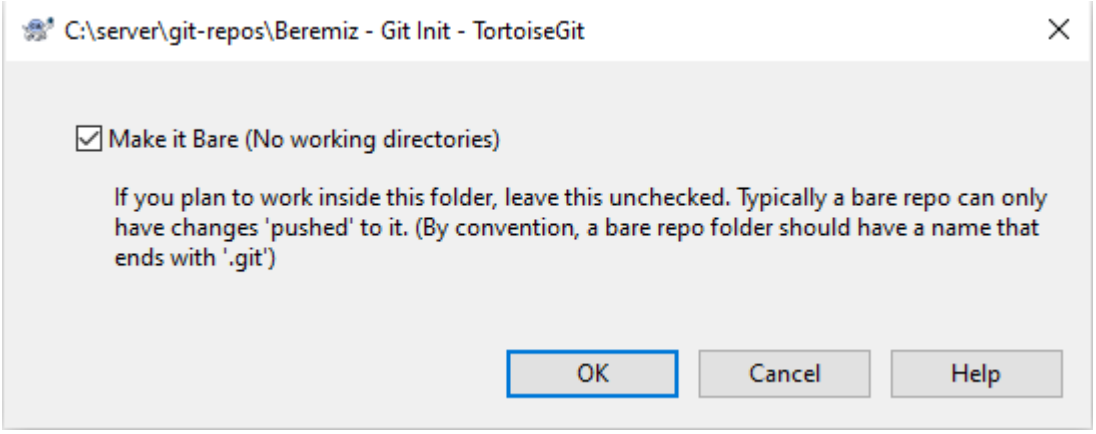


TORTOISE GIT: СОЗДАНИЕ РЕПОЗИТОРИЯ

3. Запустить процесс инициализации репозитория
   a) внутри директории `C:\git-repos\Beremiz` щелкнуть правой кнопкой мыши
   b) в появившемся контекстном меню выбрать «Git Create repository here...»



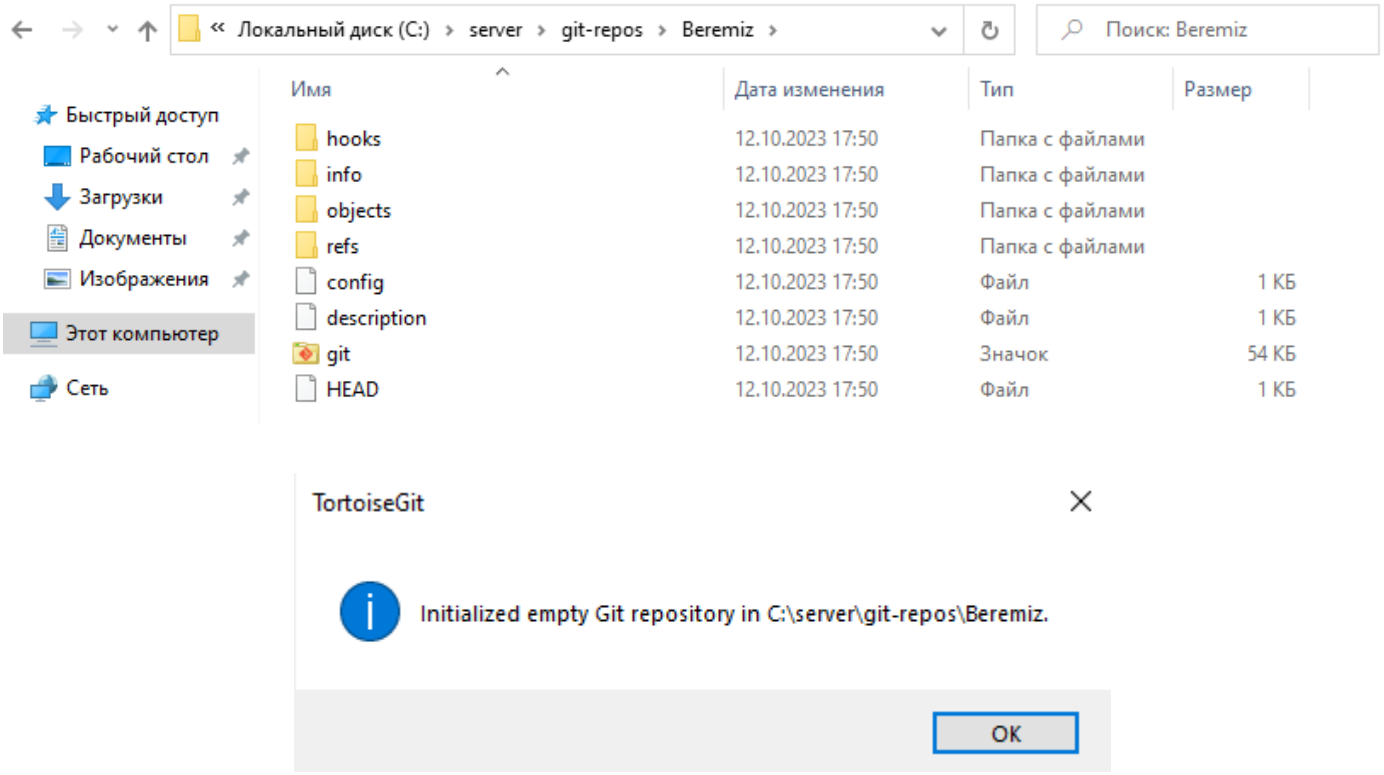
4. В появившемся диалоговом окне:
   a) включить опцию «Make it Bare»,
   b) нажать на кнопку «OK».



TORTOISE GIT: СОЗДАНИЕ РЕПОЗИТОРИЯ

В директории `C:\git-repos\Beremiz` автоматически будет создана файловая структура пустого хранилища Git.

6. В появившемся информационном окне об успешном создании пустого репозитория:
   а) нажать на кнопку «ОК».



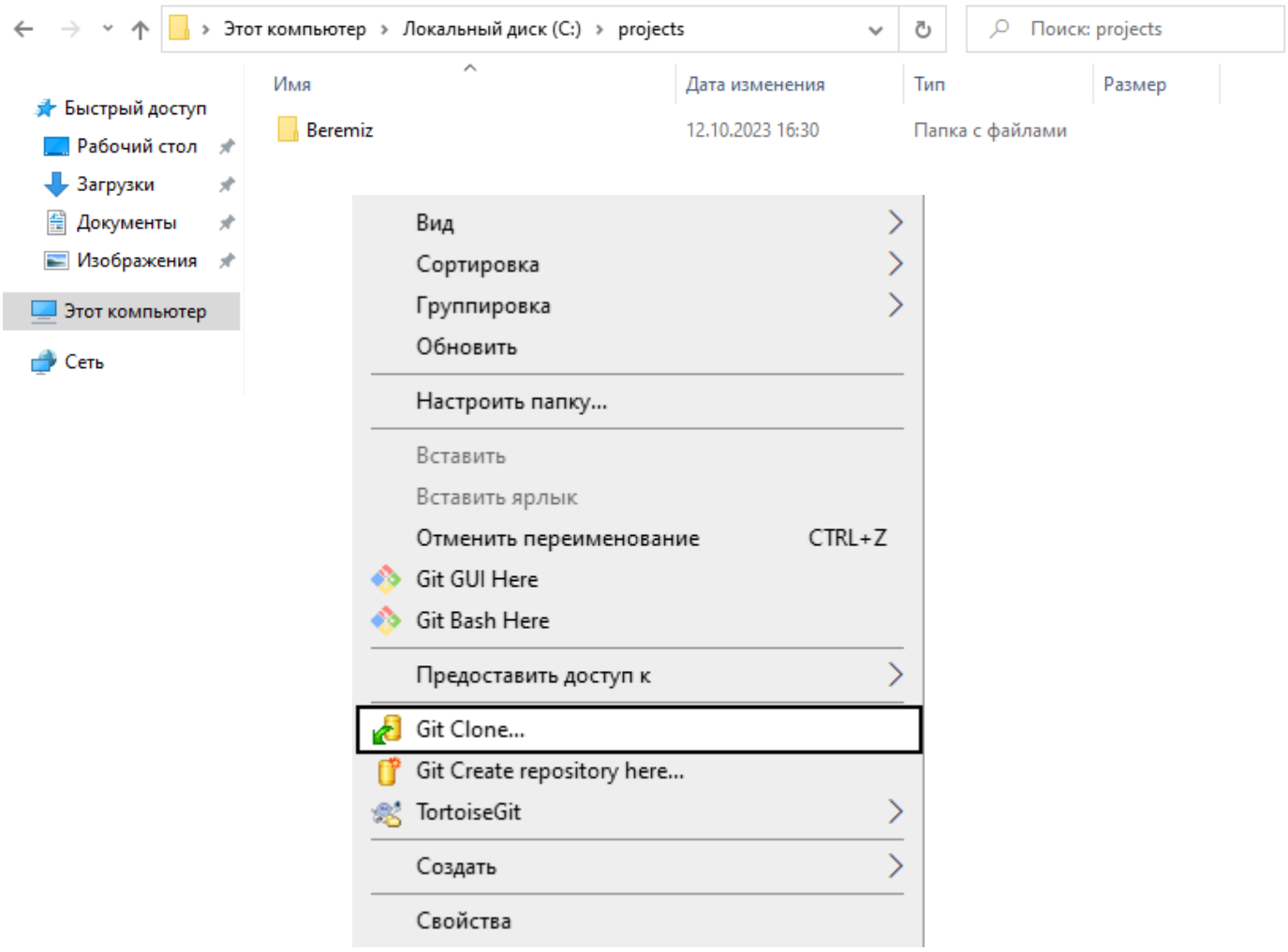
TORTOISE GIT: ИЗВЛЕЧЕНИЕ РАБОЧЕЙ КОПИИ

Пример

Задача:

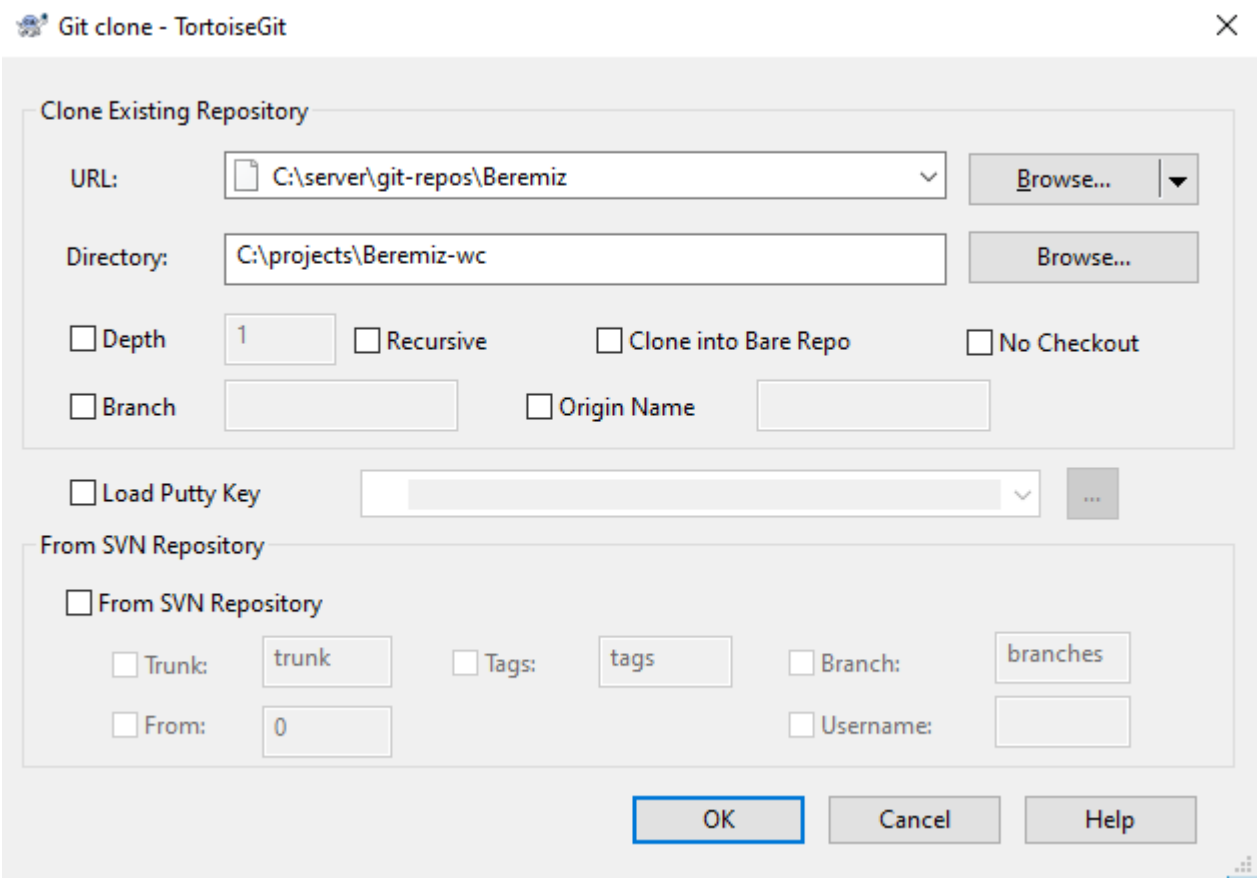
- Имеется проект «Beremiz» (с исходным кодом) в директории `C:\projects\Beremiz`
- Ранее был создан пустой репозиторий в директории `C:\server\git-repos\Beremiz`
- Создать рабочую копию для исходного проекта в директории `C:\projects\Beremiz-wc`
  где, суффикс -wc будет означать — рабочая копия (work copy)

1. Внутри директории `C:\projects`:
   a) на свободном месте щелкнуть правой клавишей мыши,
   b) в появившемся контекстном меню выбрать «Git clone...».



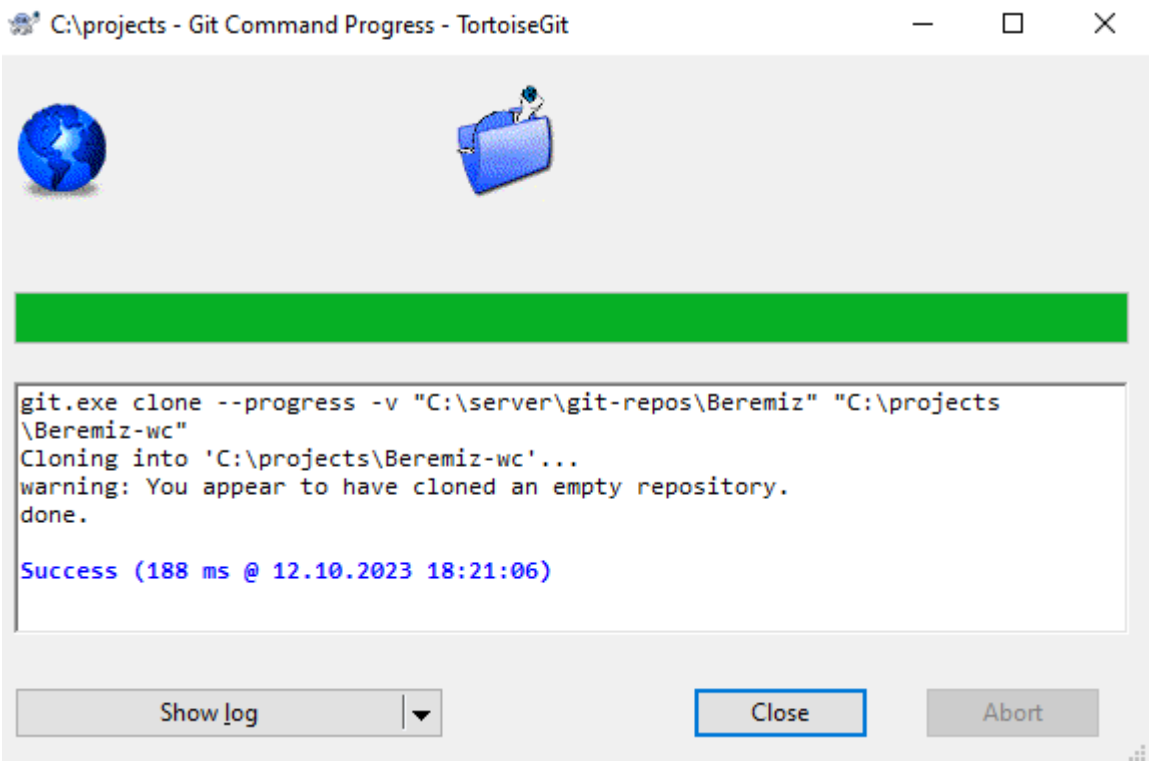
TORTOISE GIT: ИЗВЛЕЧЕНИЕ РАБОЧЕЙ КОПИИ

2. В диалоговом окне «Git clone»:
   a) указать путь к репозиторию (URL), введя его вручную или выбрав через Browse
      `C:\server\git-repos\Beremiz`
   b) указать путь (Directory) — куда будет извлечена рабочая копия
      `C:\projects\Beremiz-wc`
   c) отключить опцию «Load Putty Key».
   в) нажать на кнопку «OK».

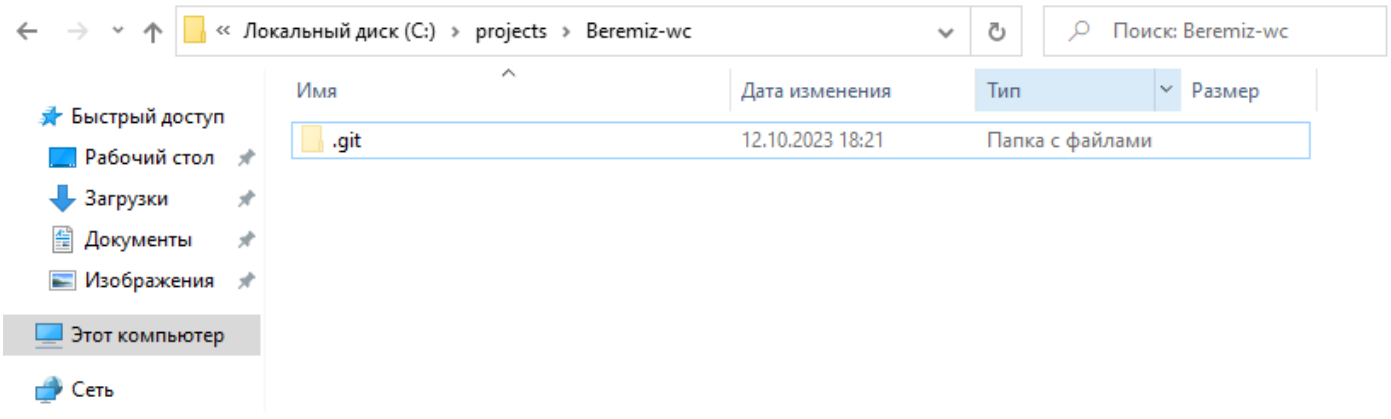


Появится диалоговое окно, визуализирующее процесс извлечения рабочей копии. Дождаться появления сообщения «Success ...».

3. В диалоговом окне визуализации процесса извлечения рабочей копии:
   а) нажать на кнопку «Close».



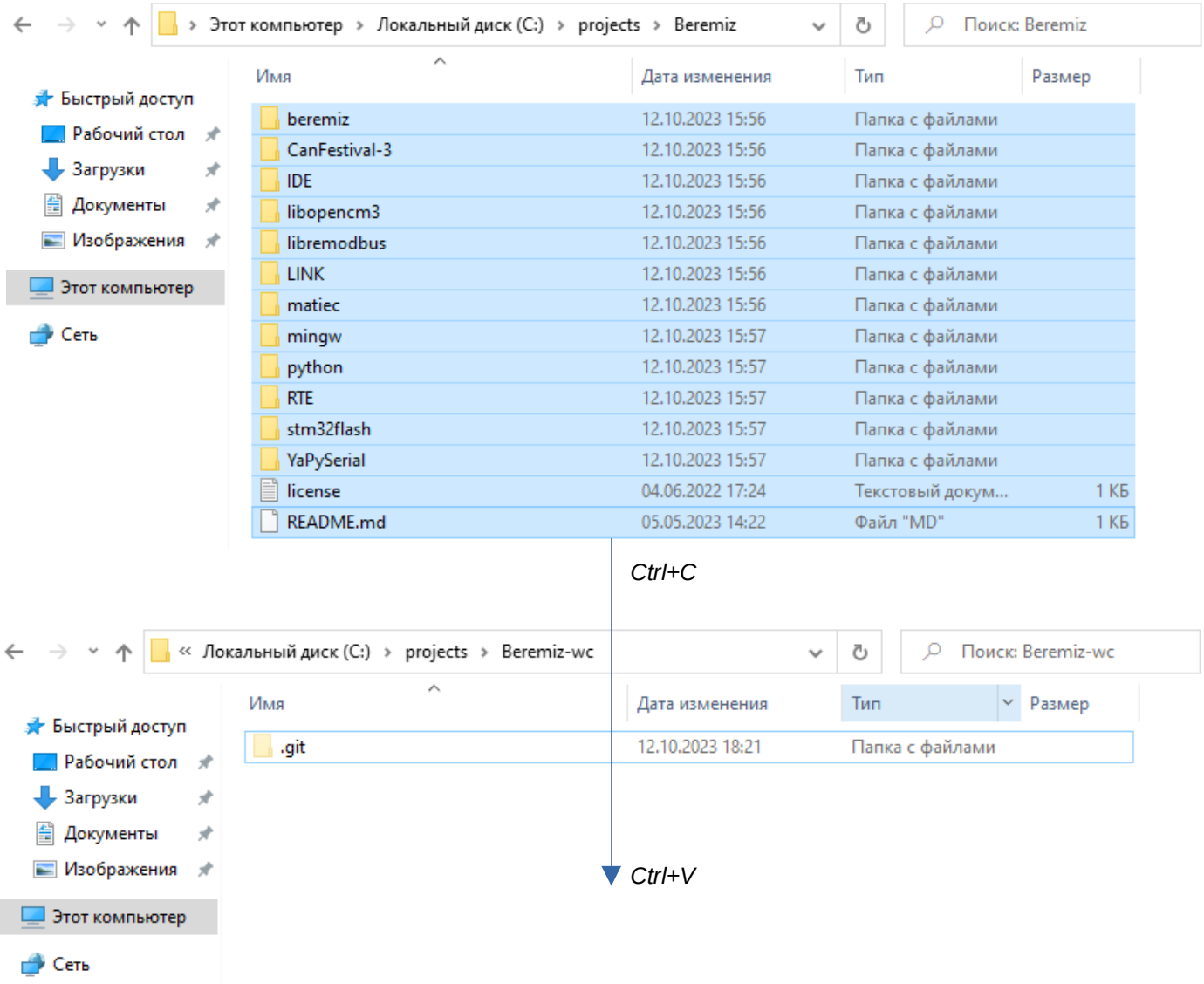
Директория рабочей копии `C:\projects\Beremiz-wc` будет содержать всего лишь одну скрытую поддиректорию «.git» (это признак рабочей копии).



TORTOISE GIT: ИЗВЛЕЧЕНИЕ РАБОЧЕЙ КОПИИ

Далее необходимо заполнить пустую рабочую копию файлами из исходного проекта.

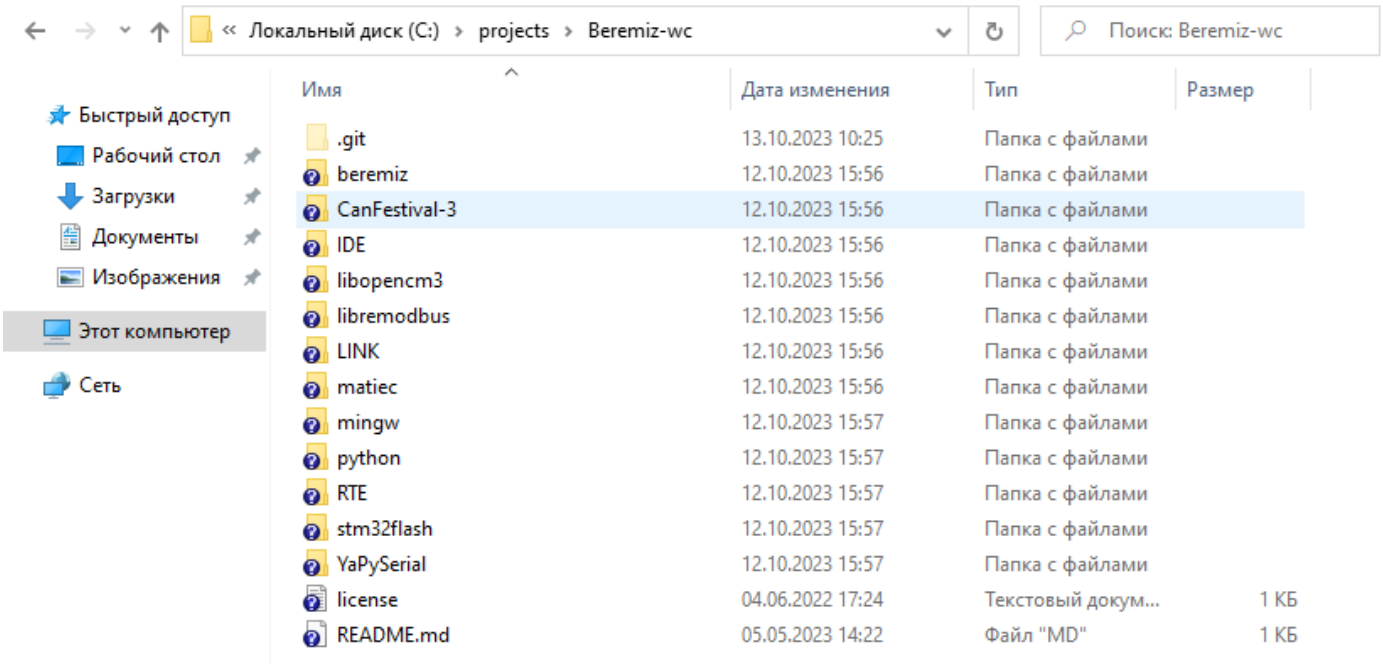
4. Скопировать все содержимое `C:\projects\Beremiz` в `C:\projects\Beremiz-wc`



TORTOISE GIT: ИЗВЛЕЧЕНИЕ РАБОЧЕЙ КОПИИ

Рабочая копия извлечена и заполнена исходным кодом проекта.

На каждом добавленном файле и директории появился ярлык со знаком «?», который означает, что это новый незафиксированный элемент (файл, директория).



Далее с этой рабочей копией можно работать в штатном режиме:

- вносить изменения в проект
- фиксировать изменения
- синхронизировать изменения с ремозиторием
- и пр.

TORTOISE GIT: ФИКСИРОВАНИЕ ИЗМЕНЕНИЙ

Пример

Задача:

- Имеется рабочая копия проекта «Beremiz»
  `C:\projects\Beremiz-wc`
  где, суффикс -wc — рабочая копия (work copy)

- Эта же рабочая копия является локальным репозиторием
  (в соответствии с моделью распределенных систем управления версиями)

- Имеется внешнее хранилище (репозиторий)
  `C:\server\git-repos\Beremiz`

- В эту рабочую копию были внесены изменения
  (например, добавлены файлы)

- Необходимо зафиксировать изменения:
  - в репозиторий рабочей копии (локально),
  - во внешний репозиторий.

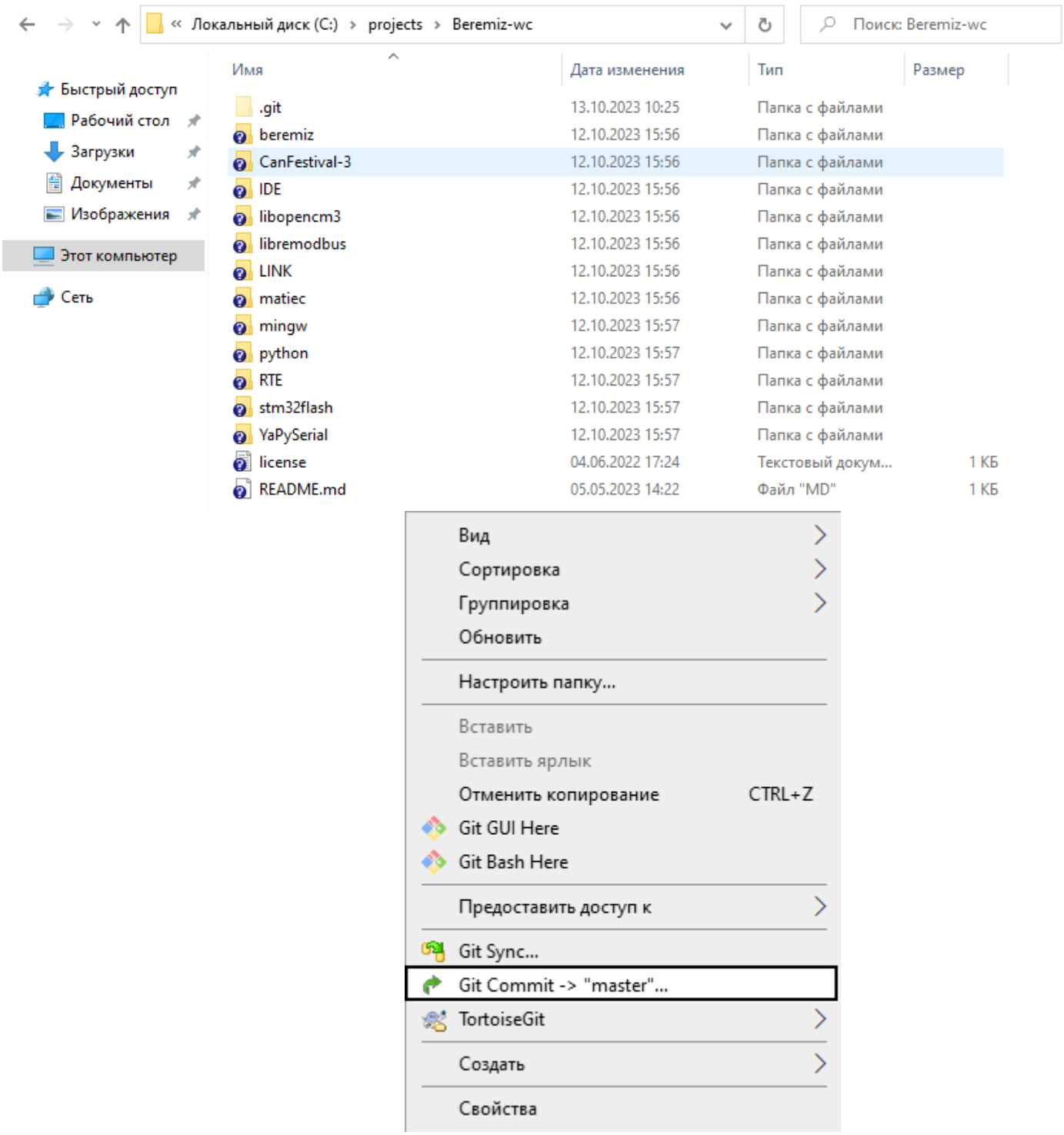
Фиксация изменний выполняется командой «commit», которая:
- выполняеят сравнение измененной версии с эталонной (предыдущей ревизией),
- проверяет наличие конфликтов,
- фиксирует фрагмент измененных данных (новая ревизия).

Так как, в распределенных системах учета версий рабочая копия является в тоже время репозиторием (локальным), то все операции фиксации изменений выполняются локально — в директорию «.git» (эталон сравнения и все ревизии хранится здесь же).

Фиксация изменений во внешний репозиторий называется — синхронизацией — и выполняется с помощью отдельной команды (sync push). Клиент TortoiseGit предоставляет механизм, позволяющий нажатием одной кнопки «Commit & Push» выполнить фиксацию и синхронизацию.

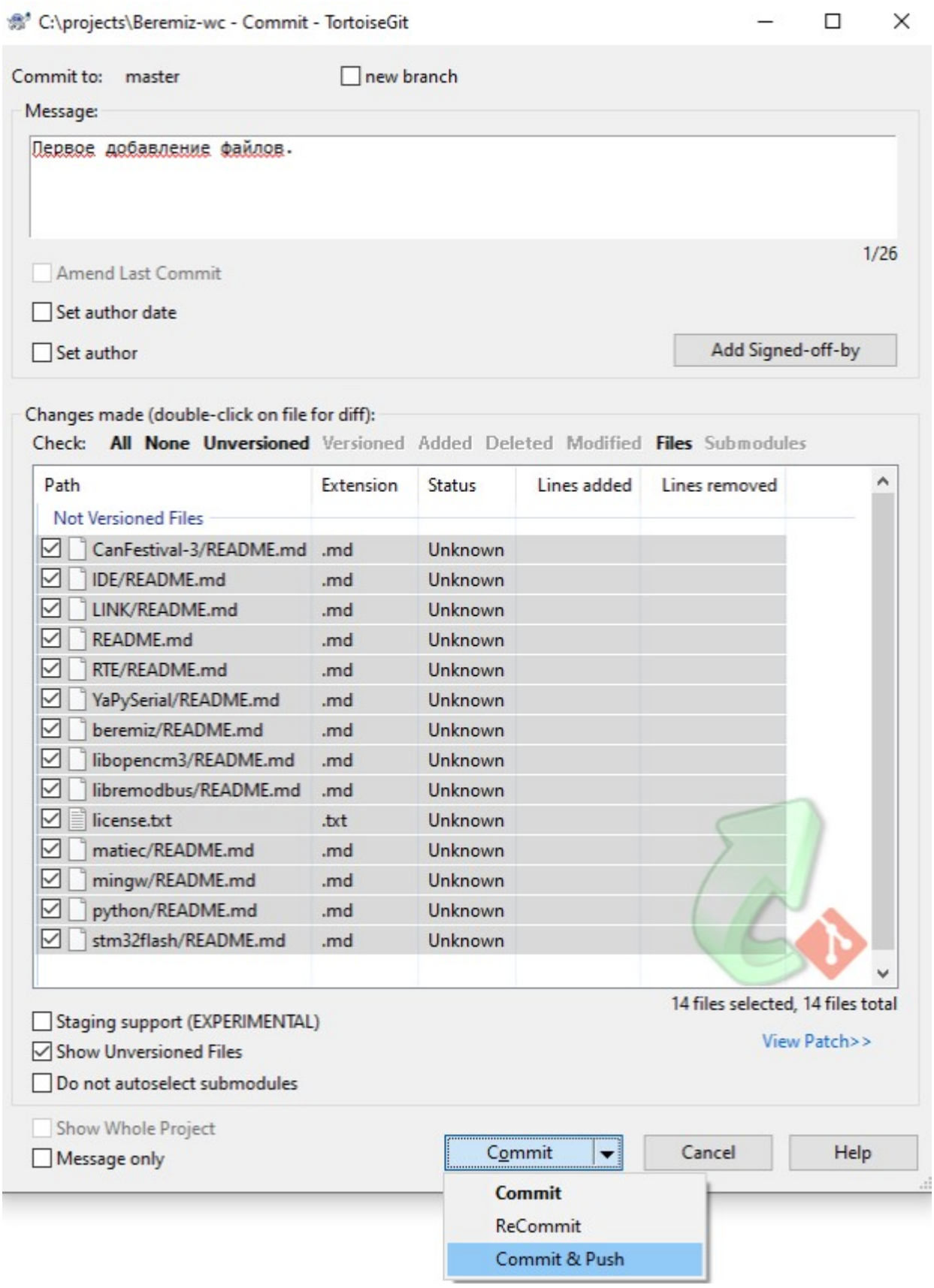
TORTOISE GIT: ФИКСИРОВАНИЕ ИЗМЕНЕНИЙ

1. Внутри директории рабочей копии `C:\projects\Beremiz-wc`:
   a) в свободной области щелкнуть правой кнопкой мыши,
   b) в появившемся контекстном меню выбрать «Git Commit → master».



TORTOISE GIT: ФИКСИРОВАНИЕ ИЗМЕНЕНИЙ

2. В диалоговом окне «Commit» :
   a) отметить все незафиксированные файлы (Not Versioned Files),
   b) ввести текст комментария (Message),
   c) в нижней части окна выбрать «Commit & Push»

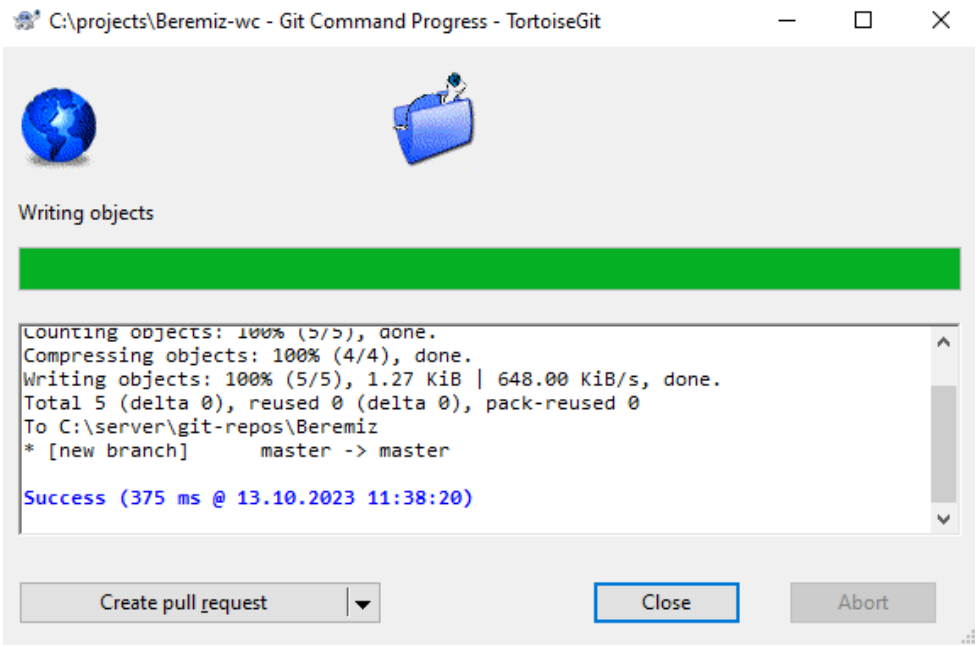


TORTOISE GIT: ФИКСИРОВАНИЕ ИЗМЕНЕНИЙ

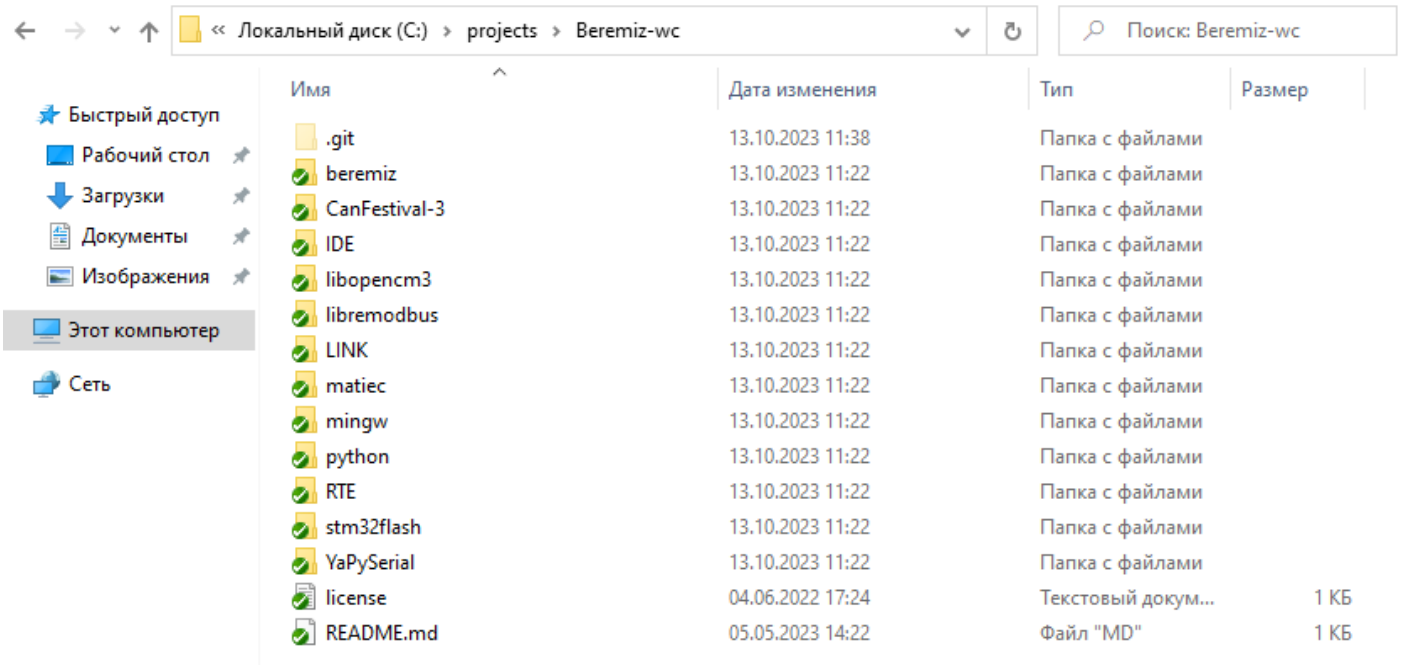
Дождаться завершения процесса фиксации изменений (локально) и передачи их во внешний репозиторий (синхронизация).

Завершение процесса визуализируется в диалоговом окне «Git Command Progress»:
a) должно появиться сообщение «Success»,
b) активироваться кнопка «Close».

3. В диалоговом окне «Git Command Progress»:
   a) нажать на кнопку «Close».



В результате, значки «?» на новыми файлами и директориями сменятся на «√» - это означает, что текущее состояние элемента соответствует эталону (номер ревизии элемента соответствует номеру ревизии эталона).



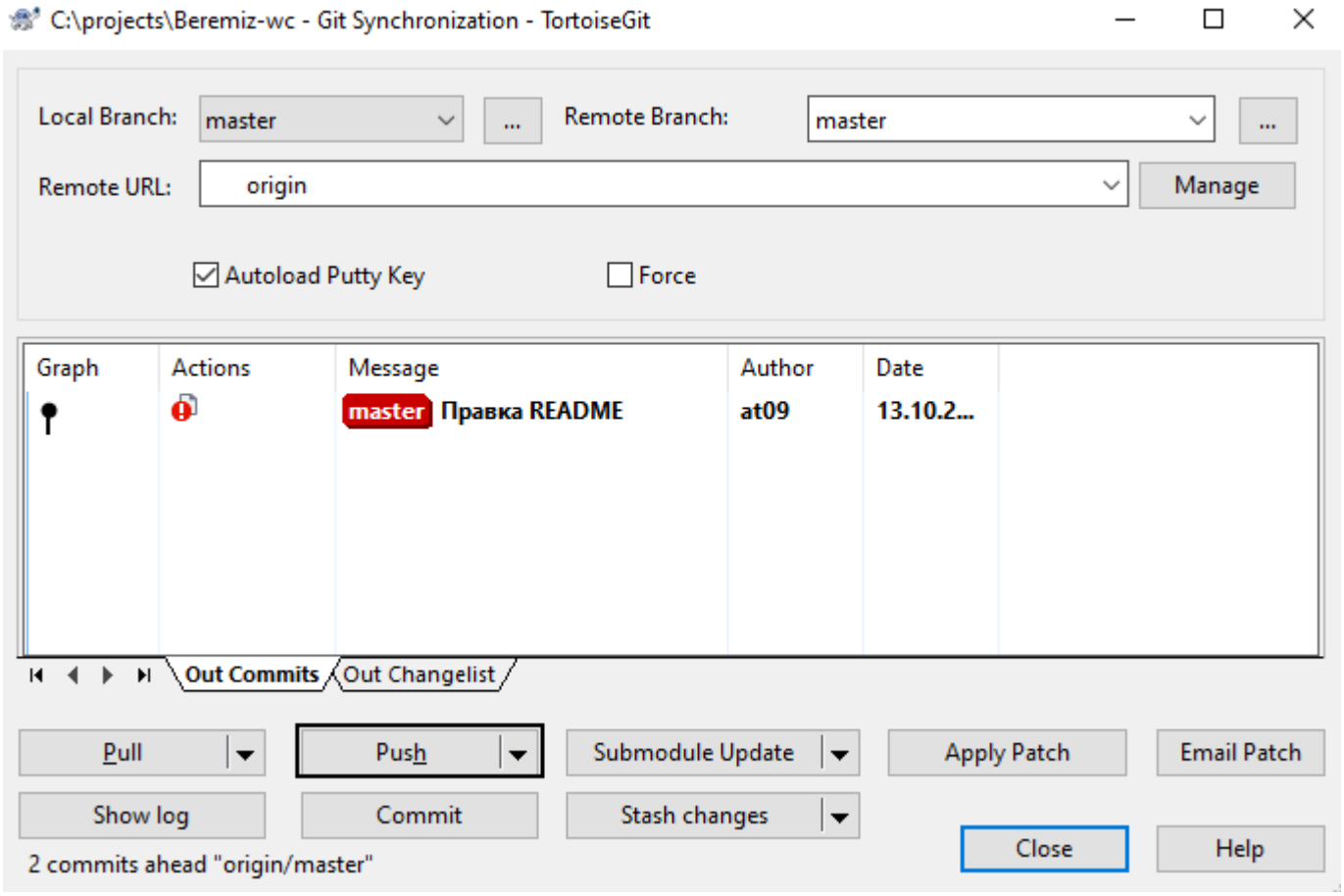
TORTOISE GIT: СИНХРОНИЗАЦИЯ С РЕПОЗИТОРИЕМ

Фиксация измерений (локальная)
commit

1. Git Commit -> master
2. Выбор незафиксированных элементов (файлов и/или директорий)
3. Commit

Отправка изменений из рабочей копии в репозиторий
sync push

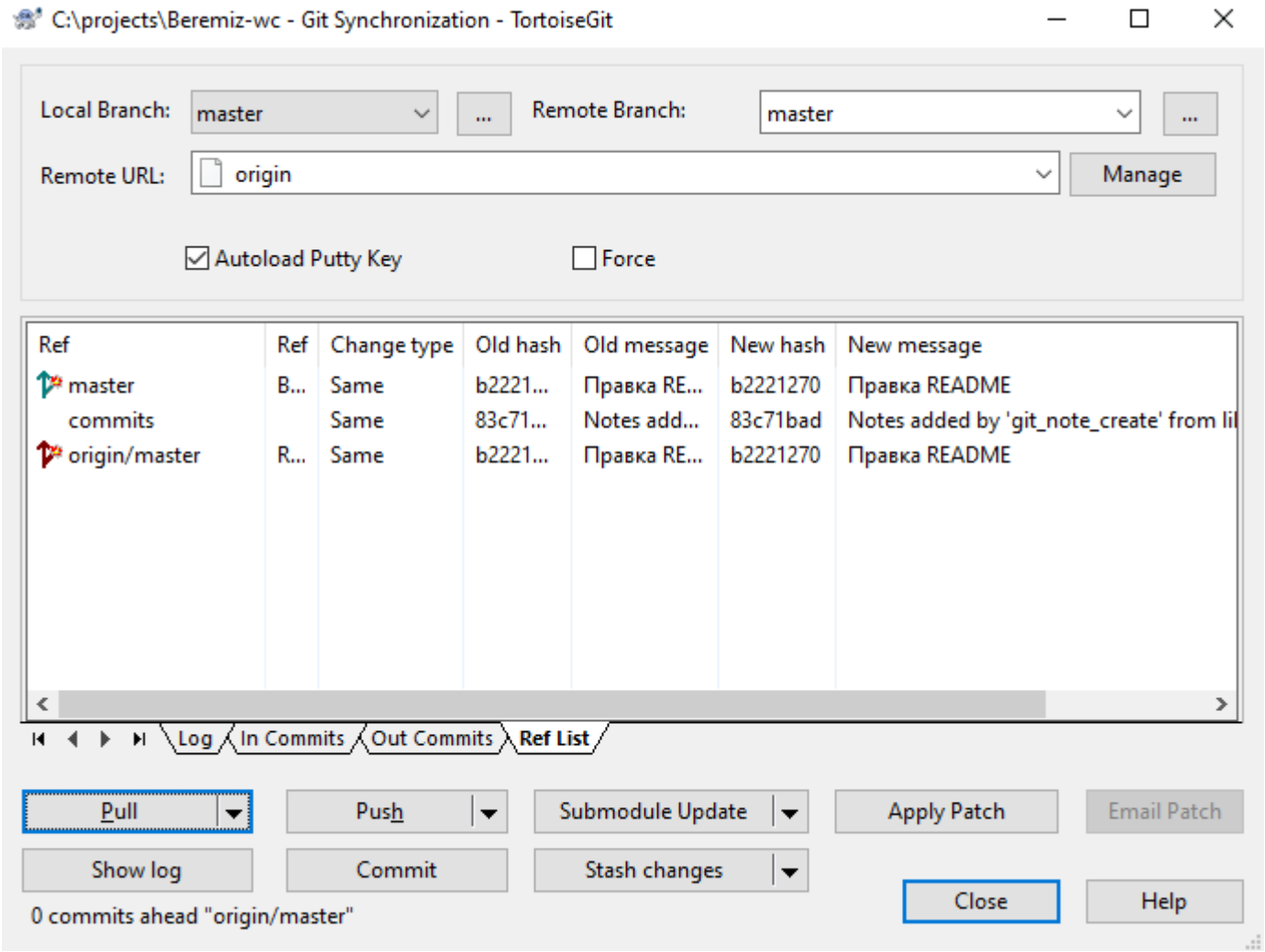
1. Git Sync,
2. выбор откуда (Local Branch) и куда (Remote Branch и Remote URL),
3. Push.



TORTOISE GIT: СИНХРОНИЗАЦИЯ С РЕПОЗИТОРИЕМ

Выгрузка изменений из репозитория в рабочую копию
sync pull

1. Git Sync,
2. выбор откуда (Local Branch) и куда (Remote Branch и Remote URL),
3. Pull.



РЕЗЕРВИРОВАНИЕ И ДУБЛИРОВАНИЕ

В отличие от SVN, в GIT нет явных (простых) способов для резервного архивирования и разархивирования репозитория целиком (в svn есть команды: backup и restore).

Поэтому, самый простой и доступный способ резервирования для Git — это использовать стороннюю программу-архиватор (например, zip).

Резервирование (минимум)
- один раз в неделю (например, в конце последнего рабочего дня недели)
- хранить последние 4 архива

Дублирование
- хранить копию резервных архивов на нескольких носителях

Пример
- целевой репозиторий
  `C:\server\git-repos\Beremiz`
- резвирование
  `C:\server\git-repos-backup\Beremiz-repos-backup-231013.zip`
- дублирование
  `D:\backup-1\Beremiz-repos-backup-231013.zip`
  `E:\backup-2\Beremiz-repos-backup-231013.zip`

Резервное копирование репозитория
пример
- репозиторий `C:\server\git-repos\Beremiz`

1. Репозиторий «Beremiz» добавить в архив «Beremiz-repos-backup-231013.zip»
   где,
   -repos-backup — суффикс, обозначающий, что это резервная копия репозитория
   -231013 — суффикс, обозначающий год-месяц-день создания копии

Восстановление репозитория из резервной копии
пример
- место расположения всех репозиториев `C:\server\git-repos\`
- резервная копия `Beremiz-repos-backup-231013.zip`

1. Архив «Beremiz-repos-backup-231013.zip» распаковать в `C:\server\git-repos\`