

## Лабораторная работа № 2

# Системы контроля версий



### Цель работы

1. Изучить на практике понятия и компоненты систем контроля версий (СКВ), приемы работы с ними.
2. Освоить специализированное ПО и распространенный сервис для работы с распределенной СКВ Git — TortoiseGit и GitHub.com.


### Общие указания к выполнению лабораторной работы

1. Система контроля версий Git представляет собой набор программ командной строки. Доступ к ним можно получить из терминала (в \*nix) или из специальной консольной оболочки Git Bash (в Windows). Однако, лабораторная работа ориентирована на применение графической надстройки TortoiseGit (аналог в Linux — RabbitVCS). TortoiseGit работает не как отдельная программа, а встраивается в контекстные меню «Проводника» Windows. Вместе с Git для Windows поставляется также программа gitk (Git GUI) — она гораздо менее популярна и пользоваться ей для ЛР не следует.
2. Задания необходимо читать внимательно и полностью. Благодаря тому, что Git является распределенной СКВ, резервную копию локального хранилища можно сделать простым копированием или архивацией.
3. В отчет можно включать снимки экрана, сообщения и комментарии по своему усмотрению с тем, чтобы было удобно пояснять сделанное, опираясь на отчет.

### Задание на лабораторную работу

1. Отработать навыки использования хранилища на локальной машине.
  - 1.1. Настроить Git, указав имя и e-mail разработчика для подписи commit-ов.   
*Указание.* Диалог настроек вызывается пунктом *TortoiseGit* → *Settings* контекстного меню любого каталога, нужная вкладка называется «Git». Задавать следует глобальные настройки (для всех хранилищ), установив переключатель «Global».
  - 1.2. Создать хранилище для учебного проекта. 
  - 1.3. Совершить несколько commit-ов.
    - 1.3.1. Скопировать `sdt.h` в каталог хранилища и создать файл `main.cpp` со включением `sdt.h` и пустой функцией `main()`.

### 1.3.2. Добавить в программу ввод двух целых чисел с приглашением.

*Указание № 1.* После выполнения каждого подпункта необходимо убеждаться, что программа работает, и совершать commit изменений. 

*Указание № 2.* Следите за тем, какие файлы отмечены в списке для commit-а изменений в них, — кроме `main.cpp` и, иногда, `sdt.h`, больше никаких других не нужно.

### 1.4. Предотвратить автоматическое добавление в хранилище файлов, не нуждающихся в контроле версий, — \*.o и \*.exe.

Правило об игнорировании следует помещать в файл `.gitignore` в корневом каталоге хранилища («`.gitignore` in repository root»). Этот файл также попадает под контроль версий, поэтому после создания правил требуется совершить commit изменений в файле `.gitignore`.

### 1.5. Добавить в программу вывод суммы введенных чисел и совершить commit.

### 1.6. Просмотреть историю (журнал) хранилища.

### 1.7. Просмотреть разность (diff) между пунктами истории 1.3.2 и 1.5.

## 2. Освоить передачу истории хранилища по сети.

### 2.1. Организовать общее хранилище на удаленном сервере.

#### 2.1.1. Зарегистрироваться на [GitHub](https://github.com).


#### 2.1.2. Создать пустое удаленное хранилище с любым наименованием.

*Указание.* Вопреки инструкции на GitHub, добавлять в хранилище файл `README.md` не нужно, удаленное хранилище должно быть пустым.

#### 2.1.3. Разрешить пользователям-преподавателям совершать commit-ы.

Требуется на странице хранилища выбрать «Settings» (справа), далее «Collaborators», где ввести имена пользователей, которым будет предоставлен полный доступ к хранилищу (эти имена можно узнать у лаборантов).

#### 2.1.4. Настроить локальное хранилище для синхронизации с удаленным.

Необходимо в контекстном меню каталога локального хранилища выбрать *TortoiseGit* → *Settings*, где перейти к пункту *Remote* . Достаточно ввести условное имя удаленного хранилища «origin» и его адрес, который отображается на веб-странице удаленного хранилища в разделе «Quick setup» (вариант HTTP). От загрузки сведений о ветвлениях в удаленном хранилище отказаться.

### 2.2. Передать локальное хранилище на удаленный сервер (push).

*Замечание.* Здесь и далее при взаимодействии с удаленным сервером потребуются вводить имя пользователя и пароль, с которыми выполнялась регистрация на GitHub.

2.3. Перейти к странице хранилища на GitHub (обновить её) и ознакомиться с возможностями просмотра содержимого через web-интерфейс.

2.4. Загрузить копию удаленного хранилища на локальную машину (clone). 

*Замечание.* Целью является имитация совместной работы с удаленным хранилищем. Для этого на одной машине организуются 2 локальных хранилища: созданное в пункте 1.1 (RepoA) и загруженное с удаленного сервера (RepoB).


*Указание.* Диалог «Git clone» следует вызывать из контекстного меню каталога *вне* локального хранилища. В качестве *URL* потребуется указать адрес удаленного хранилища, а в качестве *Directory* — имя каталога для нового локального хранилища.

2.5. Сымитировать параллельную работу над проектом.

2.5.1. В локальном хранилище RepoB добавить в программу печать разности введенных чисел, сделать commit и передать изменения на сервер.

2.5.2. В локальном хранилище RepoA добавить над функцией `main()` комментарий о том, что программа является учебной, сделать commit, **но не отправлять изменений на сервер.**


2.6. На странице хранилища на GitHub перейти в раздел *Commits* и ознакомиться с возможностью просмотра истории изменений через web-интерфейс.

2.7. В локальном хранилище RepoA выполнить загрузку с сервера новейших ветвлений и изменений (fetch) и просмотреть журнал хранилища. 

*Указание.* По умолчанию показывается только текущая активная ветвь (по умолчанию — `master`). Просмотреть все commit-ы во всех ветвях, в том числе в загруженной из удаленного хранилища ветви `origin/master`, нужно включить флажок «All branches» слева снизу окна журнала.


2.8. Совместить изменения в локальном хранилище с загруженными.

2.8.1. Использовать действие pull для загрузки изменений с удаленного сервера и автоматического совмещения их с имеющимися локально.

Просмотреть журнал изменений (или обновить кнопкой *Refresh*). 


*Примечание.* Фактически, при обновлении производится слияние ветвей `master` и `origin/master` — то есть, двух версий истории, существовавших удаленно и локально. При этом история стала нелинейной и появился лишний commit слияния. Иногда такое усложнение имеет смысл, но в данном случае было бы желательно сохранить историю линейной и просто перенести локальные наработки вслед за новейшими. Добьемся желаемого.

### 2.8.2. Отменить неудобный результат действия pull.

*Указание.* В журнале изменений в контекстном меню commit-а, где был добавлен комментарий (то есть, последнего перед слиянием), выбрать «Reset master to this...» и указать тип отмены «Hard». 

*Указание.* Журнал изменений не всегда обновляется автоматически, используйте кнопку *Refresh*, если изменения не появились сразу.

### 2.8.3. Выполнить перенос (rebase) локальных изменений на основу новейшего загруженного состояния проекта.

*Указание.* В журнале изменений в контекстном меню пункта, на котором находится конец ветви origin/master (прямоугольник в TortoiseGit), следует выбрать пункт «Rebase "master" onto this...» и далее нажать кнопку «Start rebase». 

2.9. Передать итоговое состояние локального хранилища RepoA на удаленный сервер, используя команду push.

2.10. Действуя аналогично п. п. 2.7 и 2.8.3, синхронизировать с удаленным локальное хранилище RepoB (в нем не хватает commit-а с комментарием).

*Замечание.* На данном этапе во всех трех хранилищах (локальных RepoA и RepoB и удаленном на GitHub) должна быть одинаковая линейная история из пяти — шести commit-ов.


## 3. Изучить действия, связанные с ветвлениями и разрешением конфликтов.

*Замечание.* Все действия выполняются в одном локальном хранилище, например, в RepoA.

3.1. Добавить в программу печать произведения чисел и совершите commit.

На данном этапе программа может быть такой:

```
1  #include "sdt.h"
2
3  // This program is just an example one under VCS.
4  int main()
5  {
6      int a, b;
7      cout << "Enter A and B: ";
8      cin  >> a >> b;
9      cout << "A + B = " << a + b << '\n'
10         << "A - B = " << a - b << '\n'
11         << "A * B = " << a * b << '\n';
12 }
```


3.2. Создать новую ветвь (branch) под названием division. из пункта истории, в котором был добавлен комментарий над main(). 

3.3. В новой ветви повторить пункт 3.1, заменив умножение делением.

*Указание.* Переключиться на ветвь можно, выбрав в контекстном меню commit-а, которым эта ветвь оканчивается, пункт «Switch/checkout to this». При создании ветви

можно сразу установить флажок «Switch to new branch». Переключаться можно только при чистом (clean) хранилище, то есть, без изменений в рабочей копии.

3.4. Переключиться обратно на ветвь `master`.

3.5. Выполнить слияние ветви `division` в ветвь `master` так, чтобы в последней оказался код для печати и произведения, и частного. 

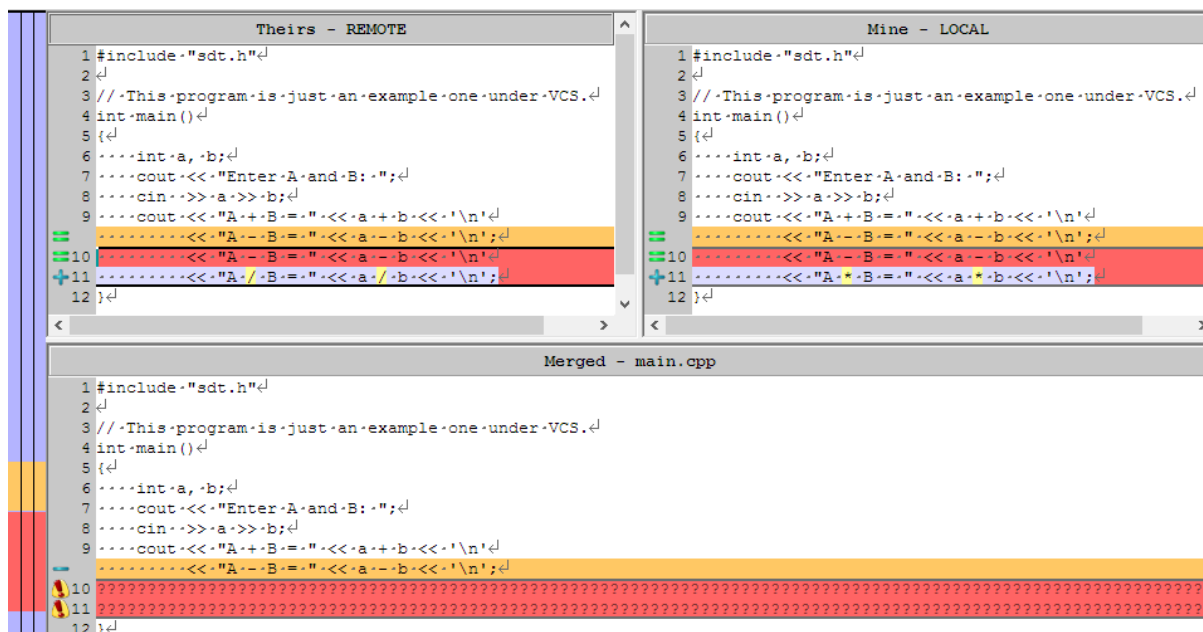
3.5.1. В журнале изменений в контекстном меню пункта-окончания ветви `division` выбрать пункт «Merge into "master"…» и начать слияние, не меняя настроек.

Действие завершится ошибкой из-за конфликта (conflict): в файле `main.cpp` строка 10 изменена в обоих `commit`-ах одинаково, а строка 11 — по-разному, и СКВ не может автоматически выбрать «правильный» вариант. Требуется вручную указать, какие строки должны войти в итоговую версию файла.

3.5.2. Приступить к разрешению конфликта.

*Указание.* Следует нажать кнопку *Resolve* (снизу), а затем выбрав пункт *Edit conflicts* из контекстного меню `main.cpp`. 

*Примечание.* Редактор конфликтов похож на программу для просмотра разностей между файлами: слева показывается файл в ветви, откуда делается слияние (`division`), справа — ветви, куда делается слияние (`master`), снизу — результат слияния. Знаками равенства в соседних верхних полях отмечаются не только строки, оставшиеся неизменными, но и строки с одинаковыми изменениями: здесь, в строке 10 убрана точка с запятой в конце в обеих ветвях. В нижнем поле каждый восклицательный знак обозначает отдельный конфликт. Примерный вид окна редактирования конфликтов представлен на рисунке ниже.



```
1 #include <stdio.h>
2
3 // This program is just an example one under VCS.
4 int main()
5 {
6     int a, b;
7     printf("Enter A and B: ");
8     scanf("%d %d", &a, &b);
9     printf("A + B = %d\n", a + b);
10    printf("A - B = %d\n", a - b);
11    printf("A / B = %d\n", a / b);
12 }
```

### 3.5.3. Разрешить конфликты:

- в качестве строки 10 предпочесть строку 10 любой ветви;
- на место строки 11 вставить 2 строки: строку с печатью произведения и строку с печатью частного;
- лишнюю точку с запятой на строке 11 поля-результата удалить.

*Указание.* Переносить строки из той или иной версии в итоговую можно, выбирая конфликтные блоки в левом или правом верхнем поле (простым щелчком левой кнопкой мыши) и пользуясь контекстным меню. Например, «Use this text block» переносит выбранный блок в поле-результат; пункт «Use text block from 'mine' before 'theirs'» переносит в результат 2 строки: сначала из правого блока, потом из левого (одну под другой). На рисунке ниже показан возможный верный результат.

```
1 #include <stdio.h>
2
3 // This program is just an example one under VCS.
4 int main()
5 {
6     int a, b;
7     printf("Enter A and B: ");
8     scanf("%d %d", &a, &b);
9     printf("A + B = " <math>a + b</math> "\n");
10    printf("A - B = " <math>a - b</math> "\n");
11    printf("A * B = " <math>a * b</math> "\n");
12    printf("A / B = " <math>a / b</math> "\n");
13}
```

### 3.5.4. Завершить процедуру разрешения конфликтов.

*Указание.* Следует нажать на кнопку «Mark as resolved», чтобы отметить файл как избавленный от конфликтов, и закрыть программу для их разрешения.

### 3.5.5. Завершить слияние ветви `division` в ветвь `master`, написав осмысленный комментарий к слиянию и совершив `commit`.

### 3.5.6. Убедиться, что программа компилируется и верно работает. Если это не так, исправить все ошибки и добиться правильной работы. Совершить `commit`.

*Замечание-указание.* Ситуация, когда после слияния программа все-таки оказывается не вполне корректной, случается на практике довольно часто. В этом случае `commit`, созданный при слиянии, оказывается логически неправильным, он не имеет ценности без последующего исправления. В Git имеется возможность

изменить (amend) уже совершенный commit, пока он не передан на сервер. Это делается при следующем commit-исправлении: следует установить флажок «Amend Last Commit» в диалоге commit — нового commit не появится, а вместо этого изменения будут приписаны предыдущему пункту истории. Можно воспользоваться данной возможностью при выполнении пункта.

#### 3.5.7. Передать все изменения всех ветвей в удаленное хранилище.

*Указание.* По умолчанию передаются только изменения текущей ветви, для передачи изменений всех ветвей следует отметить флажок «Push all branches» диалога push.

*Замечание.* В данном задании отрабатывается навык слияния ветвей, существующих только в локальном хранилище и вступивших в конфликт с ведома единственного автора. Постоянно возникают и ситуации, когда одну и ту же ветвь, но в локальном и удаленном хранилище независимо изменяют разные авторы. В этом случае действия push и rebase приведут к конфликтам. Их разрешение выполняется совершенно аналогично.

## Контрольные вопросы и задания

1. Что такое системы контроля версий (СКВ) и для решения каких задач они предназначены?
2. Объясните следующие понятия СКВ и их отношения: хранилище, commit, история, рабочая копия.
3. Что представляют собой и чем отличаются централизованные и децентрализованные СКВ? Приведите примеры СКВ каждого вида.
4. Опишите действия с СКВ при единоличной работе с хранилищем.
5. Опишите порядок работы с общим хранилищем в централизованной СКВ.
6. Что такое и зачем может быть нужна разность (diff)?
7. Что такое и зачем может быть нужно слияние (merge)?
8. Что такое конфликты (conflict) и каков процесс их разрешения (resolve)?
9. Поясните процесс синхронизации с общим хранилищем («обновления») в децентрализованной СКВ.
10. Что такое и зачем могут быть нужны ветви (branches)?
11. Объясните смысл действия rebase в СКВ Git.
12. Как и зачем можно игнорировать некоторые файлы при commit?