Контрольные вопросы:

- ☐ (5 б.) Какие концепции лежат в основе стандартной библиотеки?
- ☐ (5 б.) Зачем в проектах используются системы контроля версий?
- ☐ (5 б.) Из каких основных действий состоит взаимодействие с Git?
- ☐ (5 б.) Когда следует создавать отдельные ветки для разработки?
- ☐ (5 б.) Какие основные элементы содержатся в библиотеке chrono?

Упражнения:

В данном задании Вам предлагается повторить все шаги работы с Git, которые мы рассматривали на семинаре, а также изучить некоторые дополнительные возможности, и понять, что происходит в каждом случае. Изначально Вам достаточно иметь установленный клиент SmartGit, а также Microsoft Visual Studio с установленным расширением для работы с Git. Желательно также переключить студию на английскую локаль, т.к. перевод на русский в некоторых местах оставляет желать лучшего. Создайте проект в студии и добавьте файл исходного кода с функцией main. Далее из студии добавьте проект в систему контроля версий Git – это можно сделать через меню File – Add to Source Control. Таким образом Вы создадите локальный репозиторий. Далее нужно создать удаленный репозиторий, опубликовав Ваш локальный репозиторий на GitHub. Откройте окно командной разработки через меню View – Team Explorer. Нажмите на значок домика, далее Sync, далее Publish to GitHub. Введите логин и пароль своего GitHub аккаунта, укажите название и опубликуйте репозиторий.

Вся дальнейшая работа с Git будет осуществляться через клиент SmartGit. На семинаре я показывал Вам команду clone в учебных целях. Однако в данной ситуации в ней нет смысла, т.к. на компьютере у Вас уже имеется локальный репозиторий, созданный студией. Откройте меню Repository – Add or Create. Введите путь к папке с локальным репозиторием .git. Закрывать студию и что-либо удалять с компьютера Вам не нужно. Далее нужно убедиться, что все работает. Внесите изменения в файл исходного кода через студию, после чего он должен появиться в центральной рабочей области SmartGit в состоянии modified. Выделите его, нажмите Stage, затем Commit, затем Push. Если все в порядке, на сайте github Вы увидите свой коммит. В процессе Вам может быть предложено ввести дополнительные данные. В первом случае – это логин и пароль от Вашего аккаунта на GitHub. Тут все стандартно. Во втором случае SmartGit может предложить Вам установить пароль для ветки мастер. В этом меню следует выбрать Don't use master password. Если Вы все же выбрали использование пароля для мастера, то далее его лучше отключить через меню Edit – Preferences – Authentication – Change Master Password. Если Вы указывали какой-то пароль от мастера, то вероятнее всего, указали свой пароль от аккаунта на GitHub. Ставить пароль на мастер имеет смысл только в большом коллективе разработчиков, для того, чтобы защитить актуальную рабочую версию программного обеспечения.

В упражнениях далее Вам нужно продемонстрировать использование некоторых команд Git. В качестве решения Вы должны прислать мне ссылку на Ваш репозиторий, в котором я буду смотреть на историю изменений.

- ☐ (25 б.) Внесите несколько различных изменений в файл исходного кода. В SmartGit выберите файл и откройте Index Editor. Там Вы увидите три колонки. Правая – это Working Tree, в ней отражены изменения в Вашем файле. Центральная – это Index, в ней указаны те изменения, которые отмечены как Staged. Левая – это HEAD, в ней показано состояние файла в репозитории. Используя команды « и » перенесите часть (но не все) изменений из Working Tree в Index. Нажмите Save и закройте Index Editor. Вы должны увидеть, что Ваш файл находится в состоянии Staged Modified. Сделайте коммит. В него попадут только те изменения, которые Вы перенесли в индекс. Сделайте второй коммит для оставшихся изменений. Подобная техника удобна, когда Вы сделали много разных изменений, которые неудобно объединять одним коммитом. Сделайте Push. Зайдите на GitHub и проверьте состояние файла. Затем на сайте внесите новое изменение в файл и сделайте коммит. Вернитесь в SmartGit и выполните команду Pull, в выпадающем окне оставьте режим Rebase. Убедитесь, что изменения с удаленного репозитория подтянулись в Ваш локальный репозиторий.
- ☐ (25 б.) В случае коллективной работы над проектом и/или при наличии большого количества отдельных подзадач неудобно работать непосредственно в мастере. Для разработки нового компонента следует создавать отдельную ветку, в которой будет выполняться вся работа, при этом в мастере остается актуальная рабочая версия продукта. После отладки и тестирования нового компонента ветка вливается в мастера. Создайте новую ветку. Для этого нажмите F7, укажите название ветки, далее Add Branch. Переключитесь на новую ветку двойным щелчком по ее имени в левом нижнем окне SmartGit. Сделайте в ней коммит. Переключитесь на мастера и сделайте коммит в нем. Переключитесь обратно на новую ветку. Сейчас в мастере есть новые изменения, о которых Вашей новой ветке ничего не известно. Следует влить их в новую

ветку. Для этого нажмите ПКМ на мастере (активной должна быть Ваша новая ветка!), далее выберите Merge – Create Merge-Commit. Добавьте в Вашей новой ветке еще один коммит. Переключитесь на мастера. Влейте Вашу новую ветку в мастера с помощью Merge. Посмотрите лог и убедитесь, что Вам все понятно. Сделайте Push. Далее правильно будет удалить новую ветку, т.к. работа над компонентом закончена, и он влит в мастера. При активной ветке мастера нажмите ПКМ на имени Вашей новой ветки – Delete, поставьте все галки в появившемся окне и удалите ветку. Выставление галок позволяет удалить ветку и из удаленного репозитория тоже, если Вы делали Push Вашей ветки. В случае, если Вы не закончили работу, другой разработчик сможет взять Вашу ветку из удаленного репозитория и закончить работу за Вас. Теперь повторите весь описанный выше процесс, но на шаге вливания новых изменений из мастера в Вашу ветку используйте команду Rebase HEAD to. Посмотрите лог и убедитесь, что Вам все понятно. Возможно, в логе будет отображаться только одна ветка. Тогда для отображения полного лога в окне со списком веток следует установить галки напротив необходимых веток. Определите отличия в использовании этих двух способов и опишите их в комментариях. Дополнительно советую прочитать эту статью о Merge и Rebase.

- ☐ (25 б.) При интеграции изменений из одной ветки в другую могут возникать конфликты, в конце концов никто не запретит двум разработчикам случайно или намеренно изменить одно и тоже место в коде. Смоделируйте конфликт и решите его. Для этого создайте отдельную ветку. Сделайте в ней коммит. Переключитесь на мастера и сделайте в нем коммит, содержащий другое изменение того же места в коде, что и в новой ветке. Влейте изменения из новой ветки в мастера. Возникнет конфликт. Двойной щелчок по имени файла с конфликтом и Вы увидите окно с тремя колонками. В левой изменения другого разработчика, т.е. мастер. В правой – Ваши изменения в новой ветке. В центре то, что будет в итоге. Используя стрелочки « и » выберите нужное изменение и перенесите его в центр. Далее нажмите Save, закройте решатель конфликтов и сделайте коммит. Посмотрите лог и убедитесь, что в нем Вам все понятно. Выполните Push.
- ☐ (25 б.) Доработайте таймер с семинара так, чтобы можно было приостанавливать и возобновлять замеры.