

Технический Университет Молдовы
Кафедра Автоматики и Информационных Технологий

Лабораторная работа №1
По MIDPS

**Тема: Version Control Systems и способ установки
сервера**

Выполнила:

Проверил:

ст. гр. TI-155
Архирий Ольга
Гожин Виктор

Кишинев 2017

Цель работы:

Изучить Version Control Systems (git || bitbucket || mercurial || svn)

Задания:

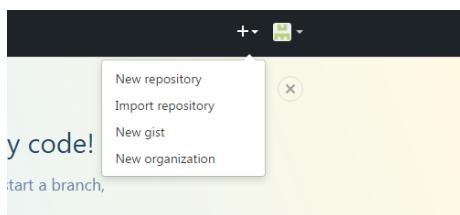
- Инициализация нового репозитория
- Конфигурация VCS
- Создание веток (не менее двух)
- Коммит на обеих ветках (не менее одного на каждой)
- Настройка ветки наблюдения за удалённой веткой (ex. Github, Bitbucket or custom server)
- Вернуть ветку к предыдущему коммиту
- Временное сохранение изменений без коммита
- Использование файла .gitignore
- Слияние двух веток
- Решение конфликта двух веток
- Усвоение команд git

Выполнение заданий:

Для изучения технологии git мы используем **GitHub** — крупнейший веб-сервис для хостинга IT-проектов и их совместной разработки. Основан на системе контроля версий Git.

Репозиторий представляет собой хранилище — место, где хранятся и поддерживаются какие-либо данные. Чаще всего данные в репозитории хранятся в виде файлов, доступных для дальнейшего распространения по сети.

Создать репозиторий можно непосредственно на сайте **GitHub**, клонируя его затем на свой локальный компьютер. Создаётся репозиторий на сайте нажатием на «+» в правом верхнем углу страницы. Клонировать репозиторий с помощью команды **git clone**.



```
User@User-MINGW64 /bin
$ git clone https://github.com/olgaar/gh1.git
Cloning into 'gh1'...
remote: Counting objects: 6, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 6 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (6/6), done.
```

Также создать репозиторий можно непосредственно через командную строку (**Git Bash** в нашем случае). Сделаем это командой **git init**, после создания пустой папки.

```

User@User- MINGW64 /d/huhg/git/bin <master>
$ mkdir hello

User@User- MINGW64 /d/huhg/git/bin <master>
$ cd hello

User@User- MINGW64 /d/huhg/git/bin/hello <master>
$ touch hello.html

User@User- MINGW64 /d/huhg/git/bin/hello <master>
$ git init
Initialized empty Git repository in D:/huhg/Git/bin/hello/.git/

```

Перед созданием нового коммита, нас попросят вписать свой e-mail и логин.

```

User@User- MINGW64 /d/huhg/git/bin/hello <master>
$ git add hello.html

User@User- MINGW64 /d/huhg/git/bin/hello <master>
$ git commit -m "First Commit"

*** Please tell me who you are.

Run

  git config --global user.email "you@example.com"
  git config --global user.name "Your Name"

to set your account's default identity.
Omit --global to set the identity only in this repository.

fatal: empty ident name (for <<null>>) not allowed

User@User- MINGW64 /d/huhg/git/bin/hello <master>
$ git config --global user.email "olga.arhiri@gmail.com"

User@User- MINGW64 /d/huhg/git/bin/hello <master>
$ git config --global user.name "olgaar"

User@User- MINGW64 /d/huhg/git/bin/hello <master>
$ git commit -m "First commit"
[master (root-commit) d42cf54] First commit
1 file changed, 1 insertion(+)
create mode 100644 hello.html

```

Ветка в Git'e — это просто легковесный подвижный указатель на один из коммитов.

Ветка по умолчанию в Git'e называется «master». Когда мы создаем коммиты на начальном этапе, нам дана ветка `master`, указывающая на последний сделанный коммит. При каждом новом коммите она сдвигается вперед автоматически.

Состояние ветки можно проверить командой **git status**. Рекомендуется добавлять в репозиторий файлы README.md (служит для комментариев пользователя) и .gitignore (указывает git игнорировать определенные файлы).

Для файла **.gitignore** существуют определённые правила синтаксиса:

- Одна строчка - одно правило,
- Пустые строки игнорируются,
- Комментарии доступны через решётку(#) в начале строки,
- Символ "/" в начале строки указывает, что правило применяется только к файлам и папкам, которые располагаются в той же папке, что и сам файл .gitignore,
- Доступно использовать спецсимволы: звёздочка(*) заменяет любое количество символов(ноль или больше), вопрос (?) заменяет от нуля до одного символа — можно размещать в любом месте правила.
- Две звёздочки (**) используются для указания любого количества поддиректорий.
- Восклицательный знак(!) в начале строки означает инвертирование правила, необходим для указания исключений из правил игнорирования,

- Символ "\" используется для экранирования спецсимволов, например, чтобы игнорировать файл с именем "!readme!.txt", нужно написать такое правило: "`\\!readme!.txt`",
- Для игнорирования всей директории, правило должно оканчиваться на слэш(/), в противном случае правило считается именем файла.

```

User@User- MINGW64 /d/huhg/git/bin/hello <master>
$ git status
On branch master
nothing to commit, working tree clean

User@User- MINGW64 /d/huhg/git/bin/hello <master>
$ touch README.md

User@User- MINGW64 /d/huhg/git/bin/hello <master>
$ touch .gitignore

User@User- MINGW64 /d/huhg/git/bin/hello <master>
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        .gitignore
        README.md

nothing added to commit but untracked files present (use "git add" to track)

User@User- MINGW64 /d/huhg/git/bin/hello <master>
$ git commit -m "Second commit"
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        .gitignore
        README.md

nothing added to commit but untracked files present

User@User- MINGW64 /d/huhg/git/bin/hello <master>
$ git add .gitignore

User@User- MINGW64 /d/huhg/git/bin/hello <master>
$ git add README.md

User@User- MINGW64 /d/huhg/git/bin/hello <master>
$ git commit -m "Second commit"
[master e16431e] Second commit
 2 files changed, 38 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 README.md

User@User- MINGW64 /d/huhg/git/bin/hello <master>
$ git status
On branch master
nothing to commit, working tree clean

```

На примере выше показано, что просто добавленный файл будет не отслеживаемым, помощью команды **git add** мы отслеживаем его и индексируем в проект. Так сохраняются временные изменения без коммита.

Мы можем также «спрятать» файлы, которые пока не хотим коммитить. Для этого используется команда **git stash**. Теперь наш рабочий каталог чист, а спрятанные файлы ждут в стеке. Мы можем перемещаться по веткам, делать коммиты, а затем вернуться и добавить файлы из стека снова в рабочий каталог. **git stash list** позволяет просмотреть стек, Если мы хотим применить одну из старых записей, можем сделать это, указав её имя так: **git stash apply stash@{2}**. Если не указывать ничего, Git будет подразумевать, что вы хотите применить последнюю спрятанную работу.

```
MINGW64:/d/Code/hello

User@User-MINGW64 /d/Code/hello (branch2)
$ git status
On branch branch2
Your branch is ahead of 'origin/branch2' by 4 commits.
(use "git push" to publish your local commits)
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   add.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        ...

User@User-MINGW64 /d/Code/hello (branch2)
$ git stash
Saved working directory and index state WIP on branch2: dd624f7 Merged master fixed conflict
HEAD is now at dd624f7 Merged master fixed conflict

User@User-MINGW64 /d/Code/hello (branch2)
$ git status
On branch branch2
Your branch is ahead of 'origin/branch2' by 4 commits.
(use "git push" to publish your local commits)
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        ...

nothing added to commit but untracked files present (use "git add" to track)

User@User-MINGW64 /d/Code/hello (branch2)
$ git stash list
stash@{0}: WIP on branch2: dd624f7 Merged master fixed conflict

User@User-MINGW64 /d/Code/hello (branch2)
$ git stash apply stash@{0}
On branch branch2
Your branch is ahead of 'origin/branch2' by 4 commits.
(use "git push" to publish your local commits)
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   add.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        ...

User@User-MINGW64 /d/Code/hello (branch2)
$ |
```

Теперь создадим новую ветку, добавим в нее файл и сделаем коммит. Перемещение между ветками происходит посредством команды **git checkout «имя ветки»**, а добавление новой ветки – **git checkout-b «имя ветки»**

```

User@User- MINGW64 /d/huhg/git/bin/hello/MIDPS/lab#1 <master>
$ git checkout -b branch1
M
hello.html
Switched to a new branch 'branch1'

User@User- MINGW64 /d/huhg/git/bin/hello/MIDPS/lab#1 <branch1>
$ git status
On branch branch1
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   ../../hello.html

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        ../../...

no changes added to commit (use "git add" and/or "git commit -a")

User@User- MINGW64 /d/huhg/git/bin/hello/MIDPS/lab#1 <branch1>
$ touch 2.txt

User@User- MINGW64 /d/huhg/git/bin/hello/MIDPS/lab#1 <branch1>
$ git add 2.txt

User@User- MINGW64 /d/huhg/git/bin/hello/MIDPS/lab#1 <branch1>
$ git commit -m "Added 2.txt"
[branch1 4b66f80] Added 2.txt
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 MIDPS/Lab#1/2.txt

```

Проделаем то же самое и со второй веткой.

Чтобы продемонстрировать работу файла **.gitignore**, откроем его через текстовый редактор. Добавим все необходимые данные и сохраним изменения. Рассмотрим строку «*.a», которая игнорирует файлы с расширением «.a». Создаём такой файл, пытаемся добавить его в проект и получаем отказ.

```

User@User- MINGW64 /d/Code/hello (branch2)
$ touch r.a

User@User- MINGW64 /d/Code/hello (branch2)
$ git add r.a
The following paths are ignored by one of your .gitignore
r.a
Use -f if you really want to add them.

```

Для просмотра коммитов можно использовать команду **git log**.

```

User@User- MINGW64 /d/huhg/git/bin/hello/MIDPS/lab#1 <branch2>
$ git log --pretty=oneline
e325b9123f83c7adc49479bfc4b9a57b931e03e4 Changed 3.txt
4ecd1500ac2bf73daf9ac7c263a0106e7bb1cc58 Added 3.txt
4b66f80f8255f6661046fa5e7e347f181fd7d242 Added 2.txt
8bd662ff61476aa22ffaf402f392959d20ad4020 Third commit
e16431edf49b8a80486d80afb8935004330ca785 Second commit
d42cf54b45ca881ab34a62abdd951f38d4246977 First commit

```

Существует также возможность вернуться к предыдущему коммиту. Используем безопасный способ – файлы и изменения не удалятся, лишь останутся непроиндексированными. Воспользуемся **git reset --soft HEAD^**.

```

User@User- MINGW64 /d/Code/hello/MIDPS/Lab#1 (branch2)
$ git reset --soft HEAD^

User@User- MINGW64 /d/Code/hello/MIDPS/Lab#1 (branch2)
$ git status
On branch branch2
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   3.txt
        new file:   4.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        ../..

```

Слияние веток производится командой **git merge**. Команда используется для слияния одной или нескольких веток в текущую. Затем она устанавливает указатель текущей ветки на результирующий коммит.

```

User@User- MINGW64 /d/Code/hello/MIDPS/Lab#1 (branch2)
$ git merge branch1
Merge made by the 'recursive' strategy.

```

Теперь мы можем отправить ветку на Github. Делается это командой **push**.

```

User@User- MINGW64 /d/Code/hello/MIDPS/Lab#1 (branch2)
$ git push -u origin branch2
Counting objects: 23, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (16/16), done.
Writing objects: 100% (23/23), 1.98 KiB | 0 bytes/s, done.
Total 23 (delta 6), reused 0 (delta 0)
remote: Resolving deltas: 100% (6/6), done.
Branch branch2 set up to track remote branch branch2 from origin.
To https://github.com/olgaar/hello.git
 * [new branch]      branch2 -> branch2

User@User- MINGW64 /d/Code/hello/MIDPS/Lab#1 (branch2)
$

```

Теперь необходимо создать конфликт. Для этого сливаем ветку master в branch2, возвращаемся в master, создаём новый файл hello.html и коммитим. Теперь в ветке branch2 содержимое файла:

```

<html>
<head>
  <!-- no style -->
</head>
<body>
  <h1>Hello, World! Life is great!</h1>
</body>
</html>

```

А в ветке master у нас записано лишь:

hello, bye

При попытке слияния возникает **конфликт**.

```
User@User-MINGW64 /d/code/hello (master)
$ git checkout branch2
Your branch is ahead of 'origin/branch2' by 2 commits.
(use "git push" to publish your local commits)
Switched to branch 'branch2'

User@User-MINGW64 /d/code/hello (branch2)
$ git merge master
Auto-merging hello.html
CONFLICT (content): Merge conflict in hello.html
Automatic merge failed; fix conflicts and then commit the result.

User@User-MINGW64 /d/code/hello (branch2|MERGING)
$
```

Содержимое файла теперь такое:

```
<<<<<<< HEAD
<html>
  <head>
    <!-- no style -->
  </head>
  <body>
    <h1>Hello, World! Life is great!</h1>
  </body>
</html>
=====
hello, bye
>>>>>>> master
```

Для решения конфликта заменим содержимое файла на:

```
<html>
<head>
  <!-- no style -->
</head>
<body>
  <h1>Hello, World! Life is great!</h1>
</body>
</html>
```

Теперь проиндексируем изменения и сделаем коммит.

```
User@User-MINGW64 /d/code/hello (branch2|MERGING)
$ git add hello.html

User@User-MINGW64 /d/code/hello (branch2|MERGING)
$ git commit -m "Merged master fixed conflict"
[branch2 dd624f7] Merged master fixed conflict
```

Существует также возможность **добавления локальной ветки для того, чтобы отслеживать удалённую**. Это делается следующим образом:

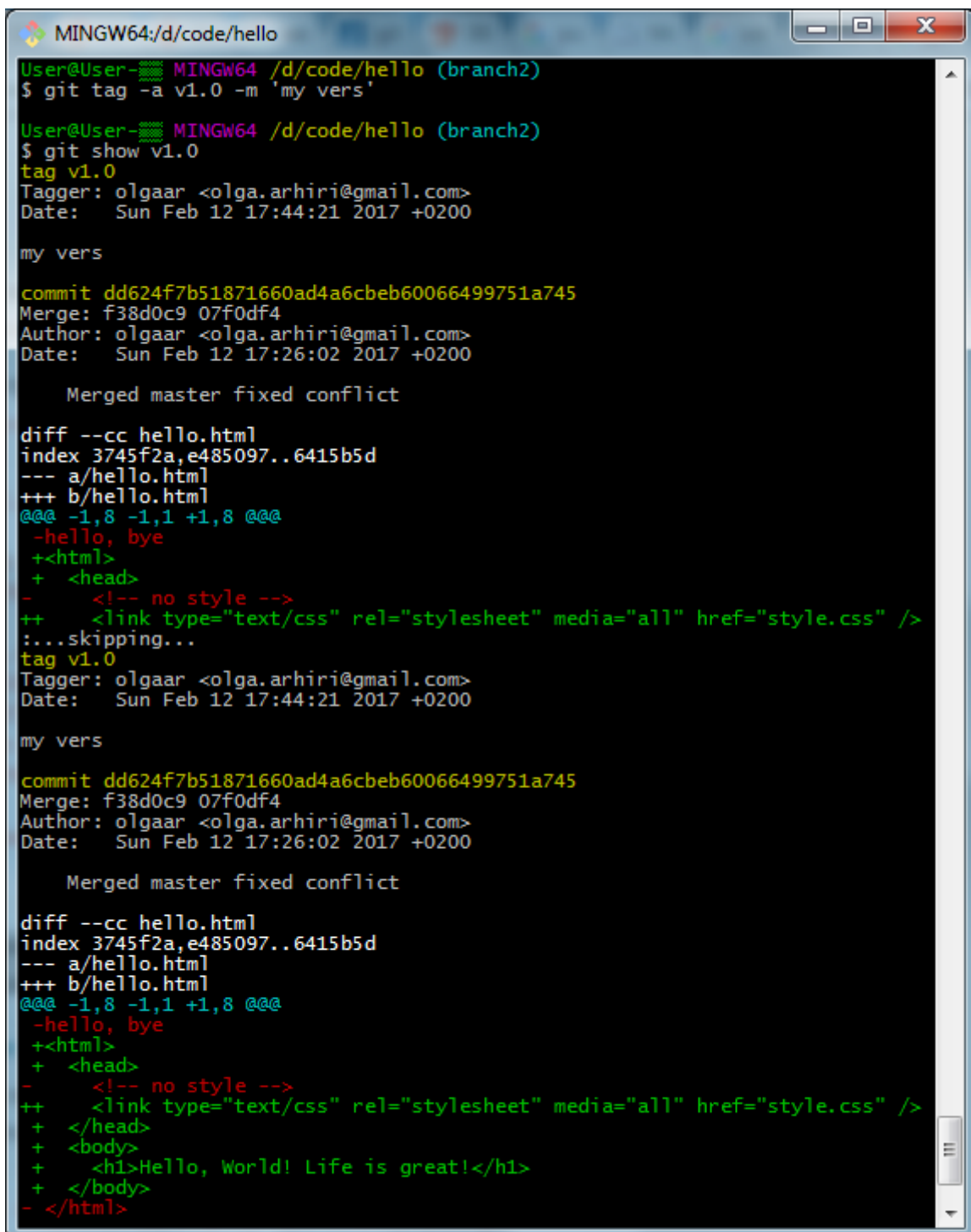

```

User@User-MINGW64 /d/code/hello (branch2)
$ git branch --track branch origin/branch2
Branch branch set up to track remote branch branch2 from origin.

User@User-MINGW64 /d/code/hello (branch2)
$ git branch -a
  branch
  branch1
  branch2
  branch3
  master
  remotes/origin/branch2
  remotes/origin/master

```

Git имеет возможность пометить (**tag**) определённые моменты в истории как важные. Как правило, этот функционал используется для отметки моментов выпуска версий (v1.0, и т.п.).



```

MINGW64:/d/code/hello
User@User-MINGW64 /d/code/hello (branch2)
$ git tag -a v1.0 -m 'my vers'

User@User-MINGW64 /d/code/hello (branch2)
$ git show v1.0
tag v1.0
Tagger: olgaar <olga.arhiri@gmail.com>
Date:   Sun Feb 12 17:44:21 2017 +0200

my vers

commit dd624f7b51871660ad4a6cbeb60066499751a745
Merge: f38d0c9 07f0df4
Author: olgaar <olga.arhiri@gmail.com>
Date:   Sun Feb 12 17:26:02 2017 +0200

    Merged master fixed conflict

diff --cc hello.html
index 3745f2a,e485097..6415b5d
--- a/hello.html
+++ b/hello.html
@@@ -1,8 -1,1 +1,8 @@@
- hello, bye
+ <html>
+ <head>
- <!-- no style -->
++ <link type="text/css" rel="stylesheet" media="all" href="style.css" />
:...skipping...
tag v1.0
Tagger: olgaar <olga.arhiri@gmail.com>
Date:   Sun Feb 12 17:44:21 2017 +0200

my vers

commit dd624f7b51871660ad4a6cbeb60066499751a745
Merge: f38d0c9 07f0df4
Author: olgaar <olga.arhiri@gmail.com>
Date:   Sun Feb 12 17:26:02 2017 +0200

    Merged master fixed conflict

diff --cc hello.html
index 3745f2a,e485097..6415b5d
--- a/hello.html
+++ b/hello.html
@@@ -1,8 -1,1 +1,8 @@@
- hello, bye
+ <html>
+ <head>
- <!-- no style -->
++ <link type="text/css" rel="stylesheet" media="all" href="style.css" />
+ </head>
+ <body>
+ <h1>Hello, World! Life is great!</h1>
+ </body>
- </html>

```

На примере выше показано создание метки и вызов информации по ней. Команда **git show** показывает информацию о выставившем метку, дату отметки коммита и аннотирующее сообщение перед информацией о коммите.

Выводы:

В ходе данной лабораторной работы я убедилась, что Git — надежный универсальный многоцелевой инструмент, чья чрезвычайная гибкость делает его надёжным инструментом в руках разработчика. С помощью данной системы контроля версий программисты могут удалённо работать над одним проектом, создавая новые файлы, скачивая последние доработки коллег, работая на разных ветках. Данная технология позволяет координировать работу всех участников разработки, это очень мощная структура, облегчающая работу путём возвращения к более ранним версиям, слияния веток, избирательной загрузки файлов (.gitignore). Мы также познакомились с сервисом Github, позволяющим создавать открытые репозитории бесплатно, лишь посредством регистрации. В ходе данной работы я пользовалась командной строкой – мощнейшим инструментом, позволяющим быстро перемещаться между директориями и файлами. Конечно, работа с командной строкой требует привыкания, но потратив время на её изучение, можно утверждать, что это достаточно удобный инструмент. Работать с Git было интересно также потому, что это – важнейший компонент современной разработки, используемый сейчас практически в любой отрасли. На мой взгляд, изучить его было важно и полезно.

Библиография:

Git documentation. [Электронный ресурс]. URL: <https://git-scm.com/doc>
Тур по основам Git. [Электронный ресурс]. URL: <https://githowto.com/ru>
Правила синтаксиса файла .gitignore. [Электронный ресурс]. URL:
<http://webnotes.by/docs/raznoe/pravila-sintaksisa-fayla-gitignore>