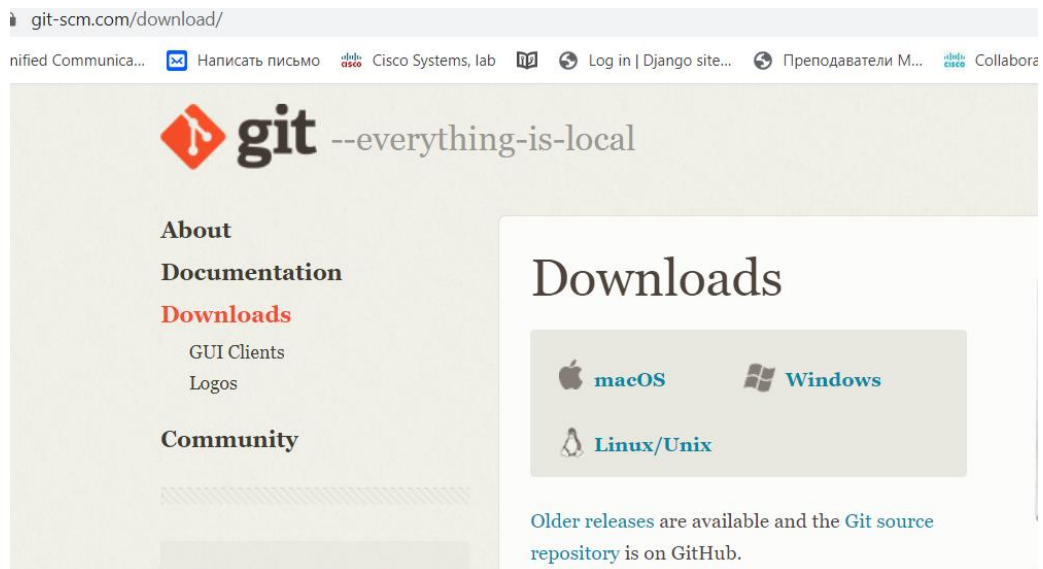


Лабораторная работа 18. Знакомство с Git.

1. Установка Git

1.1 Установите Git:



1.2 Проверьте, что git установлен. В Windows это можно сделать с помощью команды: `git --version`.

Например:

```
c:\Users\Mi>git --version
git version 2.25.1.windows.1
```

2. Подготовка к работе с git

2.1 Установка имени и электронной почты.

Выполните следующие команды, чтобы сообщить git ваше имя и электронную почту (это нужно). Если git уже установлен, можете переходить к разделу окончания строк.

```
git config --global user.name "<Your Name>"
git config --global user.email <your_email@yourmail.com>
```

Например:

```
c:\Users\Mi>git config --global user.name "Alica"
c:\Users\Mi>git config --global user.email alica_wonderland@mail.ru
```

2.2 Параметры установки окончаний строк

Для пользователей Unix/Mac:

```
git config --global core.autocrlf input  
git config --global core.safecrlf true
```

Для пользователей Windows:

```
git config --global core.autocrlf true  
git config --global core.safecrlf true
```

2.3 Установка отображения Unicode

```
git config --global core.quotepath off
```

3. Создание проекта

Цель: Научиться создавать репозиторий с нуля.

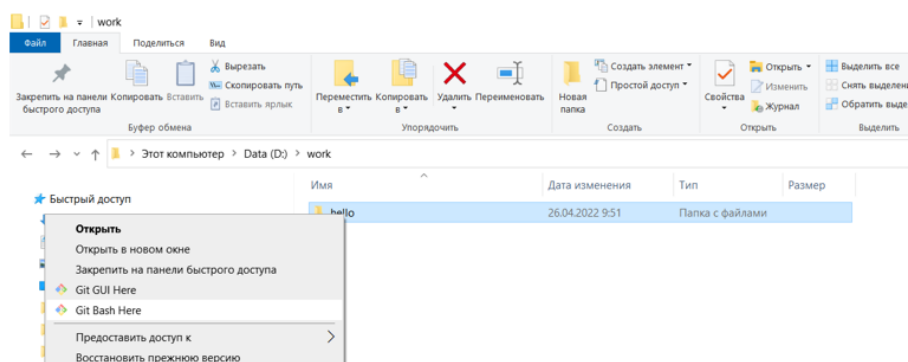
3.1 Создание папки и файла.

Создайте папку **work**, в ней создайте папку **hello**, и в нее сохраните файл с именем **hello.py** (hello.html) со следующим содержимым:

```
print ('Hello, World')
```

3.2 Создайте git репозиторий из каталога hello.

Откройте Git Bash для папки hello:



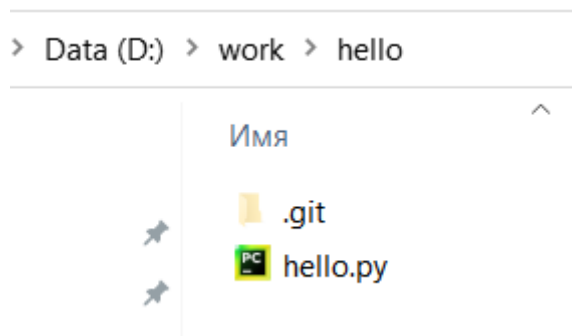
И дайте команду:

```
git init
```

Получите следующий результат:

```
mi@DESKTOP-GS5U220 MINGW64 /d/work/hello  
$ git init  
Initialized empty Git repository in D:/work/hello/.git/
```

Проверьте, что в папке hello появилась папка с именем **.git**



3.3 Добавление файла в репозиторий

Выполните команды:

```
git add hello.py
```

```
git commit -m "Мой первый коммит"
```

Результат:

```
Mi@DESKTOP-GS5U220 MINGW64 /d/work/hello (master)
$ git add hello.py

Mi@DESKTOP-GS5U220 MINGW64 /d/work/hello (master)
$ git commit --m "Мой первый коммит"
[master (root-commit) 546fd64] Мой первый коммит
1 file changed, 2 insertions(+)
create mode 100644 hello.py
```

Примечание:

Если вы опустите метку `-m` из командной строки, git перенесет вас в редактор по вашему выбору. Редактор выбирается из следующего списка (в порядке приоритета):

- переменная среды `GIT_EDITOR`
- параметр конфигурации `core.editor`
- переменная среды `VISUAL`
- переменная среды `EDITOR`

3.4. Проверка состояния

Цель: научиться проверять состояние репозитория

Для проверки состояния репозитория выполните команду:

```
git status
```

Результат:

```
$ git status
On branch master
nothing to commit, working tree clean
```

Это означает, что в репозитории хранится текущее состояние каталога и нет никаких изменений, ожидающих записи.

4. Внесение изменений

Цель: научиться отслеживать состояние рабочего каталога.

4.1. Изменение содержимого файла hello.py

Добавьте в файл hello.py новую строку:

```
Ipaddress1='10.1.1.1'
```

4.2 Проверка состояния

Проверьте состояние рабочего каталога с помощью команды:

git status

Результат:

```
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   hello.py

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

Статус показывает, что файл был изменен, но эти изменения еще не зафиксированы в репозитории. Кроме того, есть подсказка, что надо сделать дальше: использовать git add, если хотите добавить изменения в репозиторий, или git restore – если отменить изменения.

5. Индексация и коммит изменений

Цель: научиться индексировать и коммитить изменения в репозиторий.

5.1. Добавление изменений

Добавьте изменения и проверьте состояние с помощью команд:

git add hello.py

git status

```
$ git add hello.py
mi@DESKTOP-GS5U220 MINGW64 /d/work/hello (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   hello.py
```

Изменения в файле hello.py были проиндексированы, т.е. git знает об изменении, но оно пока не записано в репозиторий. Последующий коммит будет включать в себя проиндексированные изменения. В случае же, если решите не коммитить изменения, то команда состояния напомним вам о том, что с помощью команды git restore можно снять индексацию изменений.

Выполните второй коммит и посмотрите статус:

```
$ git commit --m "Мой 2й коммит - добавление ipaddress1"
[master dfb4499] Мой 2й коммит - добавление ipaddress1
1 file changed, 1 insertion(+)

Mi@DESKTOP-GS5U220 MINGW64 /d/work/hello (master)
$ git status
On branch master
nothing to commit, working tree clean
```

5.2 Добавьте два изменения и два отдельных коммита.

Git фокусируется на изменениях в файле, а не самом файле. Когда вы осуществляете команду `git add file`, вы не говорите git добавить файл в репозиторий. Скорее вы говорите, что git надо отметить текущее состояние файла, коммит которого будет произведен позже.

Добавьте новое изменение, файле `hello.py` добавьте строку `sw_name="SW1"`.

Сейчас в файле три строки:

```
print ('hello,python')
```

```
ipaddress1='10.1.1.1'
```

```
sw_name='SW-1'
```

Посмотрите статус, потом добавьте изменение в git, но пока не выполняйте коммит, посмотрите статус снова:

```
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   hello.py

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

Mi@DESKTOP-GS5U220 MINGW64 /d/work/hello (master)
$ git add .

Mi@DESKTOP-GS5U220 MINGW64 /d/work/hello (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   hello.py
```

Удалите первую строку (команду `print`) и посмотрите статус git:

```
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   hello.py

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   hello.py
```

Обратите внимание на то, что `hello.py` указан дважды в состоянии. Первое изменение (добавление новой строки) проиндексировано и готово к коммиту. Второе изменение (удаление строки с командой `print`) является непроиндексированным. Если сделать коммит сейчас, то последнее изменение не сохранится в репозитории.

Выполните команды:

```
git commit -m "Добавление новой строки"
```

```
git status
```

Получите результат:

```
$ git commit -m "Добавление новой строки"
[master b39e4ec] Добавление новой строки
1 file changed, 1 insertion(+)

mi@DESKTOP-GS5U220 MINGW64 /d/work/hello (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   hello.py

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

Состояние команды говорит о том, что `hello.html` имеет еще незафиксированные изменения, но их уже нет в staging зоне.

Добавьте второе изменение в staging зону, посмотрите статус и выполните коммит. Посмотрите снова статус.

Результаты:

```

$ git add hello.py

mi@DESKTOP-GS5U220 MINGW64 /d/work/hello (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   hello.py

mi@DESKTOP-GS5U220 MINGW64 /d/work/hello (master)
$ git commit -m "Удаление команды print"
[master c0f3ec3] Удаление команды print
1 file changed, 1 insertion(+), 1 deletion(-)

mi@DESKTOP-GS5U220 MINGW64 /d/work/hello (master)
$ git status
On branch master
nothing to commit, working tree clean

```

6. Логи

Цель. Научиться просматривать историю проекта -получать список произведенных изменений можно с помощью команды git log.

6.1. Просмотр логов

Дайте команду git log и посмотрите результаты (для завершения вывода нажмите 'q'):

```

$ git log
commit c0f3ec3c440a8b6db3a04842b24a6f56e580a79e (HEAD -> master)
Author: Alica <alica_wonderland@mail.ru>
Date:   Tue Apr 26 10:46:22 2022 +0300

    Удаление команды print

commit b39e4ec20a42c12c891e4f09d4626a228f021359
Author: Alica <alica_wonderland@mail.ru>
Date:   Tue Apr 26 10:38:44 2022 +0300

    Добавление новой строки

commit dfb44996f5d65b2f83e4ecffcbb8adee53a4f974
Author: Alica <alica_wonderland@mail.ru>
Date:   Tue Apr 26 10:23:25 2022 +0300

    Мой 2й коммит - добавление ipaddress1

commit 546fd643eb1e008228e2dbe761a34a30d5583449
Author: Alica <alica_wonderland@mail.ru>
Date:   Tue Apr 26 10:02:41 2022 +0300

    Мой первый коммит

```

Получили список всех сделанных коммитов.

6.2. Получения однострочного вывода логов

Используйте команду:

`git log --pretty=oneline`

Получите результат:

```
$ git log --pretty=oneline
c0f3ec3c440a8b6db3a04842b24a6f56e580a79e (HEAD -> master) Удаление команды print
b39e4ec20a42c12c891e4f09d4626a228f021359 Добавление новой строки
dfb44996f5d65b2f83e4ecffcb8adee53a4f974 Мой 2й коммит - добавление ipAddress1
546fd643eb1e008228e2dbe761a34a30d5583449 Мой первый коммит
```

6.3. Выборка логов.

Можно сделать выборку по логам, используя разные критерии, попробуйте:

```
$ git log --pretty=oneline --max-count=2
c0f3ec3c440a8b6db3a04842b24a6f56e580a79e (HEAD -> master) Удаление команды print
b39e4ec20a42c12c891e4f09d4626a228f021359 Добавление новой строки

Mi@DESKTOP-GS5U220 MINGW64 /d/work/hello (master)
$ git log --pretty=oneline --since '5 minutes ago'

Mi@DESKTOP-GS5U220 MINGW64 /d/work/hello (master)
$ git log --pretty=oneline --until '5 minutes ago'
c0f3ec3c440a8b6db3a04842b24a6f56e580a79e (HEAD -> master) Удаление команды print
b39e4ec20a42c12c891e4f09d4626a228f021359 Добавление новой строки
dfb44996f5d65b2f83e4ecffcb8adee53a4f974 Мой 2й коммит - добавление ipAddress1
546fd643eb1e008228e2dbe761a34a30d5583449 Мой первый коммит

Mi@DESKTOP-GS5U220 MINGW64 /d/work/hello (master)
$ git log --pretty=oneline --author=Alica
c0f3ec3c440a8b6db3a04842b24a6f56e580a79e (HEAD -> master) Удаление команды print
b39e4ec20a42c12c891e4f09d4626a228f021359 Добавление новой строки
dfb44996f5d65b2f83e4ecffcb8adee53a4f974 Мой 2й коммит - добавление ipAddress1
546fd643eb1e008228e2dbe761a34a30d5583449 Мой первый коммит

Mi@DESKTOP-GS5U220 MINGW64 /d/work/hello (master)
$ git log --pretty=oneline --author=User

Mi@DESKTOP-GS5U220 MINGW64 /d/work/hello (master)
$ git log --pretty=oneline --all
c0f3ec3c440a8b6db3a04842b24a6f56e580a79e (HEAD -> master) Удаление команды print
b39e4ec20a42c12c891e4f09d4626a228f021359 Добавление новой строки
dfb44996f5d65b2f83e4ecffcb8adee53a4f974 Мой 2й коммит - добавление ipAddress1
546fd643eb1e008228e2dbe761a34a30d5583449 Мой первый коммит
```

Посмотрим изменения, сделанные за последние 10 дней, дайте команду:

`git log --all --pretty=format:"%h %cd %s (%an)" --since='7 days ago'`

Результат:

```
$ git log --all --pretty=format:"%h %cd %s (%an)" --since='10 days ago'
c0f3ec3 Tue Apr 26 10:46:22 2022 +0300 Удаление команды print (Alica)
b39e4ec Tue Apr 26 10:38:44 2022 +0300 Добавление новой строки (Alica)
dfb4499 Tue Apr 26 10:23:25 2022 +0300 Мой 2й коммит - добавление ipAddress1 (Alica)
546fd64 Tue Apr 26 10:02:41 2022 +0300 Мой первый коммит (Alica)
```

Здесь:

`--pretty=format"..."` — определяет формат вывода.

`%h` — укороченный хэш коммита

`%cd` — дата коммитера (того, кто выполнил коммит)

%s — комментарий

%an — имя автора

Рассмотрим еще один пример форматирования ввода логов.

Дайте команду.

`git log --pretty=format:"%h %ad | %s%d [%an]" --graph --date=short`

Результат:

```
$ git log --pretty=format:"%h %ad | %s%d [%an]" --graph --date=short
* c0f3ec3 2022-04-26 | Удаление команды print (HEAD -> master) [Alica]
* b39e4ec 2022-04-26 | Добавление новой строки [Alica]
* dfb4499 2022-04-26 | Мой 2й коммит - добавление ipaddress1 [Alica]
* 546fd64 2022-04-26 | Мой первый коммит [Alica]
```

Здесь:

--pretty=format"... " — определяет формат вывода.

%h — укороченный хэш коммита

%d — дополнения коммита («головы» веток или теги)

%ad — дата коммита

%s — комментарий

%an — имя автора

--graph — отображает дерево коммитов в виде ASCII-графика

--date=short — сохраняет формат даты коротким и в приятном формате

7. Алиасы

Цель. Научиться настраивать сокращенные записи команд git (алиасы).

7.1. Создание сокращений для команд

Для часто используемых команд git удобно иметь сокращения. Например, выполните следующие команды:

`git config --global alias.co checkout`

`git config --global alias.ci commit`

`git config --global alias.st status`

`git config --global alias.br branch`

`git config --global alias.hist "log --pretty=format:'%h %ad | %s%d [%an]' --graph --date=short"`

`git config --global alias.type 'cat-file -t'`

`git config --global alias.dump 'cat-file -p'`

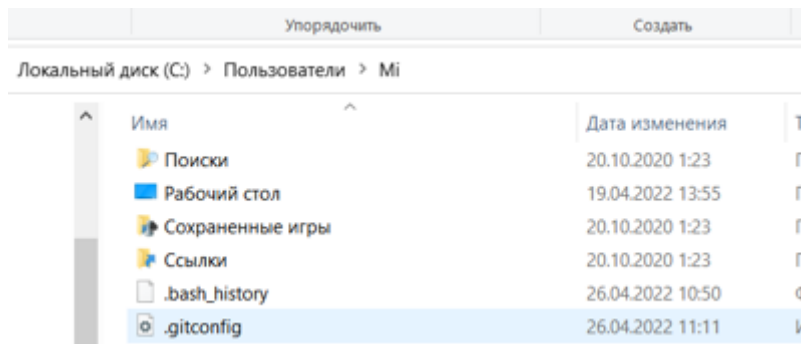
7.2. Редактирование файла .gitconfig.

Для Unix/Mac. Откройте файл .gitconfig. Он хранится в вашем \$HOME каталоге .

Добавьте алиасы:

```
[alias]
  st = status
  co = checkout
  ci = commit
  br = branch
  hist = log --pretty=format:'%h %ad | %s%d [%an]' --graph --date=short
  type = cat-file -t
  dump = cat-file -p
```

Пользователи Windows могут посмотреть ,что алиасы добавились командами.



Добавьте в файл .gitconfig еще один алиас: `hist1 = log --pretty=oneline`

Итого:

```
[alias]
  st = status
  co = checkout
  ci = commit
  br = branch
  hist = log --pretty=format:'%h %ad | %s%d [%an]' --graph --date=short
  type = cat-file -t
  dump = cat-file -p
  hist1 = log --pretty=oneline
```

8. Получение старых версий

Цель. Научиться возвращать рабочий каталог к любому предыдущему состоянию.

Команда checkout скопирует любой снимок из репозитория в рабочий каталог. П

Хеш коммита (хэш коммита, commit hash) — это специальная метка, позволяющая отличать одни коммиты от других. Хеш коммита состоит из 40 символов, в составе могут быть как буквы, так и цифры. Пример хеша коммита: d1cw4e2bd74e03ajj732assb476392ff77f24e47

8.1. Получение значения хэшей

Получим хэши предыдущих версий, для этого воспользуемся созданным алиасом hist. Дайте команду:

git log --pretty=oneline или алиас git hist1

```
$ git hist1
c0f3ec3c440a8b6db3a04842b24a6f56e580a79e (HEAD -> master) Удаление команды print
b39e4ec20a42c12c891e4f09d4626a228f021359 Добавление новой строки
dfb44996f5d65b2f83e4ecffcbb8adee53a4f974 Мой 2й коммит - добавление ipAddress1
546fd643eb1e008228e2dbe761a34a30d5583449 Мой первый коммит
```

Или команду:

git hist

Получите подобный результат:

```
$ git hist
* c0f3ec3 2022-04-26 | Удаление команды print (HEAD -> master) [Alica]
* b39e4ec 2022-04-26 | Добавление новой строки [Alica]
* dfb4499 2022-04-26 | Мой 2й коммит - добавление ipAddress1 [Alica]
* 546fd64 2022-04-26 | Мой первый коммит [Alica]
```

В первом выводе вы увидите полный хэш, а в последнем только первые 7 знаков. Их нам сейчас будет достаточно.

8.2. Посмотрите текущее содержимое файла hello.py. Можно в git bash использовать команду:

cat hello.py

```
mi@DESKTOP-GS5U220 MINGW64 /d/work/hello (master)
$ cat hello.py

ipaddress1='10.1.1.1'
sw_name='SW-1'
```

8.3. Возьмите хэш вашего первого коммита и выполните команду:

git checkout <hash>

Для хэш значений, которые приведены выше:

```
$ git checkout 546fd64
Note: switching to '546fd64'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

    git switch -c <new-branch-name>

Or undo this operation with:

    git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at 546fd64 Мой первый коммит
```

Посмотрите содержимое файла hello.py, мы вернулись к начальному состоянию файла, его содержимое состоит из одной команды print:

```
$ cat hello.py
print ('hello,python')
```

8.4. Возврат к последней версии

Выполните команды:

```
git checkout master
```

```
cat hello.html
```

Получите результат:

```
$ git checkout master
Previous HEAD position was 546fd64 Мой первый коммит
Switched to branch 'master'

mi@DESKTOP-GS5U220 MINGW64 /d/work/hello (master)
$ cat hello.py

ipaddress1='10.1.1.1'
sw_name='SW-1'
```

«master» — имя ветки по умолчанию. Переключая имена веток, вы попадаете на последнюю версию выбранной ветки.

9. Создание ветки

Цель. Научиться создавать локальную ветку в репозитории. Добавим в репозиторий новый файл и работу с ним будем выполнять в отдельной ветке

9.1. Создание нового файла

Создайте новый файл с именем my_module.py. Добавьте в него код:

```
def my_input():  
    password=input("Ведите пароль: ")  
    return password
```

9.2. Создание новой ветки и переключение на нее.

Для создания новой ветки с именем module и переключения на нее дайте команду:

git checkout -b module

```
$ git checkout -b module  
Switched to a new branch 'module'
```

Это действие можно выполнить также двумя командами:

Создание ветки: git branch module

Переключение на ветку: git checkout module.

9.3. Посмотрите статус git и добавьте файл my_module.py:

```
$ git status  
On branch module  
Untracked files:  
  (use "git add <file>..." to include in what will be committed)  
    my_module.py  
  
nothing added to commit but untracked files present (use "git add" to track)
```

Сейчас находимся на ветке module и появился неотслеживаемый в git файл my_module.py

Добавьте его и выполните коммит:

```
$ git add my_module.py  
warning: CRLF will be replaced by LF in my_module.py.  
The file will have its original line endings in your working directory  
Mi@DESKTOP-GS5U220 MINGW64 /d/work/hello (module)  
$ git commit -m "add new file my_module.py"  
[module 88f0ace] add new file my_module.py  
1 file changed, 5 insertions(+)  
create mode 100644 my_module.py  
  
Mi@DESKTOP-GS5U220 MINGW64 /d/work/hello (module)  
$ git status  
On branch module  
nothing to commit, working tree clean
```

9.4. Измените файл hello.py и выполните коммит.

Добавьте код в файл hello.py, чтобы в итоге получилось следующее:

```
from my_module import my_input
ipaddress1='10.1.1.1'
sw_name='SW-1'
password_sw=my_input()
print (f'Пароль установлен, новое значение: {password_sw}')
```

Добавьте изменения и выполните коммит:

```
$ git status
On branch module
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   hello.py

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

Mi@DESKTOP-GS5U220 MINGW64 /d/work/hello (module)
$ git add .

Mi@DESKTOP-GS5U220 MINGW64 /d/work/hello (module)
$ git commit -m "подключение модуля my_module.py"
[module dff734e] подключение модуля my_module.py
1 file changed, 3 insertions(+), 2 deletions(-)
```

Теперь есть 2 коммита в новой ветке module.

10. Навигация по веткам.

Цель. Научиться перемещаться между ветками репозитория. Теперь в нашем проекте есть две ветки, дайте команду:

```
git hist --all
```

Результат:

```
$ git hist --all
* dff734e 2022-04-26 | подключение модуля my_module.py (HEAD -> module) [Alica]
* 88f0ace 2022-04-26 | add new file my_module.py [Alica]
* c0f3ec3 2022-04-26 | Удаление команды print (master) [Alica]
* b39e4ec 2022-04-26 | Добавление новой строки [Alica]
* dfb4499 2022-04-26 | Мой 2й коммит - добавление ipaddress1 [Alica]
* 546fd64 2022-04-26 | Мой первый коммит [Alica]
```

10.1. Переключение на ветку master.

Для переключения между ветками можно использовать git checkout или git switch.

Выполните команды:

```
git checkout master
```

```
cat hello.py
```

```
$ git checkout master
Switched to branch 'master'

Mi@DESKTOP-GS5U220 MINGW64 /d/work/hello (master)
$ cat hello.py

ipaddress1='10.1.1.1'
sw_name='SW-1'
```

Мы переключились в ветку master, в ней в файле hello.py еще нет изменений.

Шаг 2. Переключитесь в ветку module и посмотрите содержимое файла hello.py в этой ветке.

Выполните команды

```
git switch module
```

```
cat hello.py
```

Результат:

```
$ git switch module
Switched to branch 'module'

Mi@DESKTOP-GS5U220 MINGW64 /d/work/hello (module)
$ cat hello.py
from my_module import my_input
ipaddress1='10.1.1.1'
sw_name='SW-1'
password_sw=my_input()
print (f'Пароль установлен, новое значение: {password_sw}')
```

11. Изменения в ветке master

11.1. Создайте новый файл в ветке master.

Создайте файл readme.txt со следующим содержимым:

This is the Hello World example for this lab.

11.2. Переключитесь в ветку master и сделайте коммит для readme.txt. Выполните команды:

```
git checkout master
```

```
$ git checkout master
Switched to branch 'master'
```

```
git add readme.txt
```

```
git commit -m "Добавлено readme.txt"
```

```
$ git add readme.txt
Mi@DESKTOP-GS5U220 MINGW64 /d/work/hello (master)
$ git commit -m "Добавлено readme.txt"
[master 58b3fee] Добавлено readme.txt
1 file changed, 1 insertion(+)
create mode 100644 readme.txt
```

12. Просмотр отличающихся веток.

Цель. Научиться просматривать отличающиеся ветки в репозитории.

Теперь в репозитории есть две отличающиеся ветки. Используйте следующую лог-команду для просмотра веток и их отличий.

git hist --all

```
$ git hist --all
* 58b3fee 2022-04-26 | Добавлено readme.txt (HEAD -> master) [Alica]
* dff734e 2022-04-26 | подключение модуля my_module.py (module) [Alica]
* 88f0ace 2022-04-26 | add new file my_module.py [Alica]
/
* c0f3ec3 2022-04-26 | Удаление команды print [Alica]
* b39e4ec 2022-04-26 | Добавление новой строки [Alica]
* dfb4499 2022-04-26 | Мой 2й коммит - добавление ipaddress1 [Alica]
* 546fd64 2022-04-26 | Мой первый коммит [Alica]
```

Здесь можно увидеть в действии --graph в git hist. Добавление опции --graph в git log вызывает построение дерева коммитов с помощью простых ASCII символов. Мы видим обе ветки (style и master), и то, что ветка master является текущей HEAD. Общим предшественником обеих веток является коммит «Added index.html».

Метка --all гарантированно означает, что мы видим все ветки. По умолчанию показывается только текущая ветка.

13. Слияние веток

Цель. Научиться сливать две отличающиеся ветки для переноса изменений обратно в одну ветку.

Слияние переносит изменения из двух веток в одну. Вернитесь к ветке module и слейте master с module.

Выполните :

git checkout module

git merge master

```
$ git checkout module
Switched to branch 'module'
Mi@DESKTOP-GS5U220 MINGW64 /d/work/hello (module)
$ git merge master
Merge made by the 'ort' strategy.
 readme.txt | 1 +
1 file changed, 1 insertion(+)
create mode 100644 readme.txt
```


git hist --all

```
$ git hist --all
* 7ad7a1e 2022-04-26 | Merge branch 'master' into module (HEAD -> module) [Alica]
|
| * 58b3fee 2022-04-26 | Добавлено readme.txt (master) [Alica]
| * dff734e 2022-04-26 | подключение модуля my_module.py [Alica]
| * 88f0ace 2022-04-26 | add new file my_module.py [Alica]
|/
* c0f3ec3 2022-04-26 | Удаление команды print [Alica]
* b39e4ec 2022-04-26 | Добавление новой строки [Alica]
* dfb4499 2022-04-26 | Мой 2й коммит - добавление ipaddress1 [Alica]
* 546fd64 2022-04-26 | Мой первый коммит [Alica]
```

Путем периодического слияния ветки master с веткой style вы можете переносить из master любые изменения и поддерживать совместимость изменений style с изменениями в основной ветке.

14. Создание конфликта

Цель. Создание конфликтующих изменений в ветке master.

14.1. Вернитесь в ветку мастер, с помощью команды:

git switch master

Добавьте в начале кода hello.py комментарий:

This is new comment

```
Mi@DESKTOP-GS5U220 MINGW64 /d/work/hello (module)
$ git checkout master
Switched to branch 'master'

Mi@DESKTOP-GS5U220 MINGW64 /d/work/hello (master)
$ cat my_module.py
cat: my_module.py: No such file or directory

Mi@DESKTOP-GS5U220 MINGW64 /d/work/hello (master)
$ cat hello.py

ipaddress1='10.1.1.1'
sw_name='SW-1'

Mi@DESKTOP-GS5U220 MINGW64 /d/work/hello (master)
$ git add .

Mi@DESKTOP-GS5U220 MINGW64 /d/work/hello (master)
$ git commit -m "Добавление комментария в hello.py, в ветке master"
[master 6e4d06d] Добавление комментария в hello.py, в ветке master
1 file changed, 1 insertion(+), 1 deletion(-)
```

14.2 Посмотрите ветки с помощью команды:

git hist --all

```
$ git hist --all
* 6e4d06d 2022-04-26 | Добавление комментария в hello.py, в ветке master (HEAD -> master) [Alica]
* 7ad7a1e 2022-04-26 | Merge branch 'master' into module (module) [Alica]
/
* 58b3fee 2022-04-26 | Добавлено readme.txt [Alica]
* dff734e 2022-04-26 | подключение модуля my_module.py [Alica]
* 88f0ace 2022-04-26 | add new file my_module.py [Alica]
/
* c0f3ec3 2022-04-26 | Удаление команды print [Alica]
* b39e4ec 2022-04-26 | Добавление новой строки [Alica]
* dfb4499 2022-04-26 | Мой 2й коммит - добавление ipAddress1 [Alica]
* 546fd64 2022-04-26 | Мой первый коммит [Alica]
```

После коммита «Добавлено readme.txt» ветка master была объединена с веткой style, но в настоящее время в master есть дополнительный коммит, который не был слит с my_module.py

15. Разрешение конфликтов

Цель. Научиться разрешать конфликты во время слияния.

15. 1. Слияние master с веткой style

Перейдите к ветке module и попытайтесь объединить ее с обновленной веткой master. Выполните команды:

git checkout module

git merge master

Результат:

```
$ git checkout module
Switched to branch 'module'

Mi@DESKTOP-GS5U220 MINGW64 /d/work/hello (module)
$ git merge master
Auto-merging hello.py
CONFLICT (content): Merge conflict in hello.py
Automatic merge failed; fix conflicts and then commit the result.
```

15.2г Откройте файл hello.py. Вы увидите следующее содержимое:

```
|<<<<<< HEAD
from my_module import my_input
=====
# This is new comment

>>>>>> master
ipAddress1='10.1.1.1'
sw_name='SW-1'
password_sw=my_input()
print (f'Пароль установлен, новое значение: {password_sw}')
```

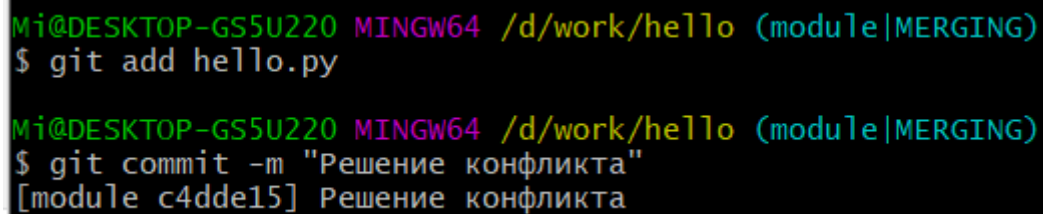
Здесь версии ветки module и master.

15.3 Решение конфликта.

Конфликт необходимо вручную разрешить конфликт. Внесите изменения в hello.py для достижения следующего результата:

```
from my_module import my_input
# This is new comment
ipaddress1='10.1.1.1'
sw_name='SW-1'
password_sw=my_input()
print (f'Пароль установлен, новое значение: {password_sw}')
```

15.4 Выполните коммит для решения конфликта.



```
Mi@DESKTOP-GS5U220 MINGW64 /d/work/hello (module|MERGING)
$ git add hello.py

Mi@DESKTOP-GS5U220 MINGW64 /d/work/hello (module|MERGING)
$ git commit -m "Решение конфликта"
[module c4dde15] Решение конфликта
```

15.5. Слияние в ветку master

Выполните команды:

git checkout master

Посмотрите содержимое файлов

git merge module

Посмотрите содержимое файлов снова

Результаты:

```

Mi@DESKTOP-GS5U220 MINGW64 /d/work/hello (module)
$ git checkout master
Switched to branch 'master'

Mi@DESKTOP-GS5U220 MINGW64 /d/work/hello (master)
$ cat my_module.py
cat: my_module.py: No such file or directory

Mi@DESKTOP-GS5U220 MINGW64 /d/work/hello (master)
$ cat hello.py
# This is new comment

ipaddress1='10.1.1.1'
sw_name='SW-1'

Mi@DESKTOP-GS5U220 MINGW64 /d/work/hello (master)
$ git merge module
Updating 6e4d06d..c4dde15
Fast-forward
 hello.py      | 4 +++-
 my_module.py | 5 +++++
 2 files changed, 8 insertions(+), 1 deletion(-)
 create mode 100644 my_module.py

Mi@DESKTOP-GS5U220 MINGW64 /d/work/hello (master)
$ cat my_module.py
def my_input():
    password=input("Ведите пароль: ")
    print(password)
my_input()

Mi@DESKTOP-GS5U220 MINGW64 /d/work/hello (master)
$ cat hello.py
from my_module import my_input
# This is new comment
ipaddress1='10.1.1.1'
sw_name='SW-1'
password_sw=my_input()
print (f'Пароль установлен, новое значение: {password_sw}')

```

Посмотрите логи:

```

$ git hist
* c4dde15 2022-04-26 | Решение конфликта (HEAD -> master, module) [Alica]
|
| * 6e4d06d 2022-04-26 | Добавление комментария в hello.py, в ветке master [Alica]
* | 7ad7a1e 2022-04-26 | Merge branch 'master' into module [Alica]
|/
| * 58b3fee 2022-04-26 | Добавлено readme.txt [Alica]
* | dff734e 2022-04-26 | подключение модуля my_module.py [Alica]
* | 88f0ace 2022-04-26 | add new file my_module.py [Alica]
|/
* c0f3ec3 2022-04-26 | Удаление команды print [Alica]
* b39e4ec 2022-04-26 | Добавление новой строки [Alica]
* dfb4499 2022-04-26 | Мой 2й коммит - добавление ipaddress1 [Alica]
* 546fd64 2022-04-26 | Мой первый коммит [Alica]

```

Теперь ветки module и master идентичны.

16. Сравнения.

16.1. Просмотр изменений, не внесенных в индекс. Внесите изменение в файл `hello.py`, добавьте две пустых строки после комментария `#`.

Выполните команду:

`git diff`

Получите результаты:

```
$ git diff
diff --git a/hello.py b/hello.py
index 144443b..c8e5ed3 100644
--- a/hello.py
+++ b/hello.py
@@ -1,5 +1,7 @@
 from my_module import my_input
 # This is new comment
+
+
 ipaddress1='10.1.1.1'
 sw_name='SW-1'
 password_sw=my_input()
```

16.2. Сравнение проиндексированных изменений с локальным репозиторием.

Если вы изменили какие-нибудь файлы в вашем рабочем каталоге и добавили один или несколько из них в индекс (с помощью `git add`), то команда `git diff` не покажет изменения в этих файлах. Чтобы показать изменения в файлах, включая файлы, добавленные в индекс, используется ключ `--cached`:

```
$ git diff --cached

Mi@DESKTOP-GS5U220 MINGW64 /d/work/hello (master)
$ git add .

Mi@DESKTOP-GS5U220 MINGW64 /d/work/hello (master)
$ git diff --cached
diff --git a/hello.py b/hello.py
index 144443b..c8e5ed3 100644
--- a/hello.py
+++ b/hello.py
@@ -1,5 +1,7 @@
 from my_module import my_input
 # This is new comment
+
+
 ipaddress1='10.1.1.1'
 sw_name='SW-1'
 password_sw=my_input()
```

Или с помощью ключа `--staged`:

```
$ git diff --staged
diff --git a/hello.py b/hello.py
index 144443b..c8e5ed3 100644
--- a/hello.py
+++ b/hello.py
@@ -1,5 +1,7 @@
 from my_module import my_input
 # This is new comment
+
+
 ipaddress1='10.1.1.1'
 sw_name='SW-1'
 password_sw=my_input()
```

16.3. Сравнение коммитов

Сравните первый и последний коммит, сначала получите значения хэшей.

```
$ git log --oneline
c4dde15 (HEAD -> master, module) Решение конфликта
6e4d06d Добавление комментария в hello.py, в ветке master
7ad7a1e Merge branch 'master' into module
58b3fee Добавлено readme.txt
dff734e подключение модуля my_module.py
88f0ace add new file my_module.py
c0f3ec3 Удаление команды print
b39e4ec Добавление новой строки
dfb4499 Мой 2й коммит - добавление ipaddress1
546fd64 Мой первый коммит
```

Теперь сравним два коммита. Для этого в качестве первого аргумента команде git diff указывается хеш первого коммита, а вторым аргументом хеш второго коммита которые сравниваем:

```

$ git diff c4dde15 546fd64
diff --git a/hello.py b/hello.py
index 144443b..f79f4df 100644
--- a/hello.py
+++ b/hello.py
@@ -1,6 +1,2 @@
-from my_module import my_input
-# This is new comment
-ipaddress1='10.1.1.1'
-sw_name='SW-1'
-password_sw=my_input()
-print (f'Пароль установлен, новое значение: {password_sw}')
\ No newline at end of file
+
+print ('hello,python')
diff --git a/my_module.py b/my_module.py
deleted file mode 100644
index fa39d9b..0000000
--- a/my_module.py
+++ /dev/null
@@ -1,5 +0,0 @@
-def my_input():
-    password=input("Ведите пароль: ")
-    print(password)
-my_input()

diff --git a/readme.txt b/readme.txt
deleted file mode 100644
index cd67c8f..0000000
--- a/readme.txt
+++ /dev/null
@@ -1 +0,0 @@
-This is the Hello World example for this lab.
\ No newline at end of file

```

Приложение

С помощью параметра `--pretty=format:""` можно указать, какие именно данные о коммите нужно выводить, определив внутри кавычек общий паттерн, используя следующие обозначения:

%H Хеш коммита

%h Сокращённый хеш коммита

%d Имена ссылок на коммит

%s Сообщение к коммиту

%an Автор

%ad Дата автора

%cn Коммитер

%cd Дата коммитера

%Cred Переключить цвет на красный

%Cgreen Переключить цвет на зелёный

%Cblue Переключить цвет на синий

%Creset Сбросить цвет