1. Изучаем Git.

Файлы для проекта “проба\_01” помещаются в папку на диске D.

Адрес D:\Git\_Doc\Project\_proba\_1

1. Вносим изменения в это файл. Потом – Git status. Он должен будет определиться Git, как измененный. Чтобы проиндексировать его – git add.
2. Вносим изменения в файл «Изучаем Git».   
   Потом git status. Показывает, что «Изучаем Git» есть и в проиндексированном списке Changes to be commited и в не проиндексированном. Changes not staged for commit.  
   Если вы выполните коммит сейчас, то файл «Изучаем Git» попадёт в коммит в том состоянии, в котором он находился, когда вы последний раз выполняли команду git add.  
   Надо выполнить опять git add, чтобы проиндексировать последнюю версию.
3. Синтаксис пробела в Git.   
   Написание в Win: Изучаем Git  
   Написание в Git: Изучаем\ Git
4. Команда git diff.   
   Важно отметить, что git diff сама по себе не показывает все изменения, сделанные с последнего коммита — только те, что ещё не проиндексированы. Такое поведение может сбивать с толку, так как если вы проиндексируете все свои изменения, то git diff ничего не вернёт.  
   Если вы хотите посмотреть, что вы проиндексировали и что войдёт в следующий коммит, вы можете выполнить   
   git diff --cached.   
   git diff --staged (В Git’е версии 1.6.1 и выше, вы также можете использовать git diff --staged, которая легче запоминается.) Эта команда сравнивает ваши индексированные изменения с последним коммитом.
5. Этапы работы с Git  
   Редактирование файла -> Индексирование (git add) -> Фиксация (git commit -m "здесь пишем комментарий")
6. git log выводит список коммитов созданных в данном репозитории в обратном хронологическом порядке. То есть самые последние коммиты показываются первыми.  
   - Один из наиболее полезных параметров — это -p, который показывает дельту (разницу/diff), привнесенную каждым коммитом. Вы также можете использовать -2, что ограничит вывод до двух последних записей  
   git log -p -2  
   - если вы хотите получить некоторую краткую статистику по каждому коммиту, вы можете использовать параметр --stat  
   git log –stat  
   параметр --stat выводит под каждым коммитом список изменённых файлов, количество изменённых файлов, а также количество добавленных и удалённых строк в этих файлах. Он также выводит сводную информацию в конце.  
   - параметры, ограничивающие по времени, такие как --since и --until, весьма полезны. Например, следующая команда выдаёт список коммитов, сделанных за последние две недели:

git log --since=2.weeks  
  
- Если у вас есть желание использовать какой-нибудь графический инструмент для визуализации истории коммитов, можно попробовать распространяемую вместе с Git’ом программу gitk  
В верхней части окна располагается история коммитов вместе с подробным графом наследников. Просмотрщик дельт в нижней половине окна отображает изменения, сделанные

выбранным коммитом. Указать коммит можно с помощью щелчка мышью.

1. **Командная строка редактора Vim** (<https://losst.ru/kak-polzovatsya-tekstovym-redaktorom-vim>)

Текстовый редактор Vim может работать в двух режимах. Это и есть его главная особенность. Первый режим, который используется по умолчанию при открытии редактора - это командный. В этом режиме вы можете вводить команды vi, а также использовать символьные клавиши для управления редактором. Второй режим - обычное редактирование текста, он работает так же как и редактирование текста в nano. Для переключения в командный режим используется клавиша Esc. Для переключения в режим редактирования - клавиша i.  
Командная строка Vim запускается в командном режиме нажатием двоеточия - ":". Здесь доступны команды для сохранения файла и выхода из редактора, настройки внешнего вида и взаимодействия с внешней оболочкой. Рассмотрим наиболее часто используемые команды редактора vim:

:w - сохранить файл;

:q - закрыть редактор;

:q! - закрыть редактор без сохранения;

:e файл - прочитать содержимое файла в позицию курсора;

:r файл - вставить в содержимое файла в следующую строку;

:r! - выполнить команду оболочки и вставить ответ в редактор;

:set переменная=значение - установить значение переменной, например, tabstop=4, или set number, с помощью этой команды можно управлять многими аспектами работы vim.

:buffers - посмотреть открытые файлы.

1. Как теперь отменить индексацию одного из двух файлов? Команда git status напомнит

вам об этом:

$ git add .

$ git status

# On branch master

# Changes to be committed:

# (use "git reset HEAD <file>..." to unstage)

#

# modified: README.txt

# modified: benchmarks.rb

#

Сразу после надписи «Changes to be commied», написано использовать git reset

HEAD <файл>... для исключения из индекса.

1. Как быстро отменить изменения, вернуть то состояние, в котором он находился во время последнего коммита (или первоначального клонирования, или какогото другого действия, после которого файл попал в рабочий каталог)? К счастью, git status говорит, как добиться и этого.  
   # 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: benchmarks.rb  
Давайте сделаем то, что написано:

$ git checkout -- benchmarks.rb

$ git status

# On branch master

# Changes to be committed:

# (use "git reset HEAD <file>..." to unstage)

#

# modified: README.txt

#

Как видите, изменения были отменены. Вы должны понимать, что это опасная команда: все сделанные вами изменения в этом файле пропали — вы просто скопировали поверх него другой файл. Никогда не используйте эту команду, если вы не полностью уверены, что этот файл вам не нужен. Если вам нужно просто сделать, чтобы он не мешался, мы рассмотрим прятание (stash) и ветвление в следующей главе; эти способы обычно более предпочтительны.

1. Если у вас есть желание пропустить этап индексирования, Git предоставляет простой способ. Добавление параметра -a в команду git commit заставляет Git автоматически индексировать каждый уже отслеживаемый на момент коммита файл, позволяя вам обойтись без git add:

$ git status

# On branch master

#

# Changes not staged for commit:

#

# modified: benchmarks.rb

#

$ git commit -a -m 'added new benchmarks'

[master 83e38c7] added new benchmarks

1 files changed, 5 insertions(+), 0 deletions(-)

Обратите внимание на то, что в данном случае перед коммитом вам не нужно выполнять git add для файла benchmarks.rb.

1. Чтобы просмотреть, какие удалённые серверы у вас уже настроены, следует выполнить команду git remote. Если вы склонировали ваш репозиторий, у вас должен отобразиться, по крайней мере, origin — это имя по умолчанию, которое Git присваивает серверу, с которого вы склонировали.  
   Чтобы посмотреть, какому URL соответствует сокращённое имя в Git, можно указать команде опцию –v.
2. Чтобы добавить новый удалённый Git-репозиторий под именем-сокращением, к которому будет проще обращаться, выполните git remote add [сокращение] [url]:

$ git remote

origin

$ git remote add pb git://github.com/paulboone/ticgit.git

$ git remote -v

origin git://github.com/schacon/ticgit.git

pb git://github.com/paulboone/ticgit.git  
  
Теперь вы можете использовать в командной строке имя pb вместо полного URL. Например, если вы хотите извлечь (fetch) всю информацию, которая есть в репозитории Павла, но нет в вашем, вы можете выполнить git fetch pb.

1. Fetch и Pull

Как вы только что узнали, для получения данных из удалённых проектов, следует выполнить:

$ git fetch [имя удал. сервера]

Данная команда связывается с указанным удалённым проектом и забирает все те данные проекта, которых у вас ещё нет. После того как вы выполнили команду, у вас должны появиться ссылки на все ветки из этого удалённого проекта. Теперь эти ветки в любой момент могут быть просмотрены или слиты. Важно отметить, что команда fetch забирает данные в ваш локальный репозиторий, но не сливает их с какими-либо вашими наработками и не модифицирует то, над чем вы работаете в данный момент. Вам необходимо вручную слить эти данные с вашими, когда вы будете готовы.  
Если у вас есть ветка, настроенная на отслеживание удалённой ветки (для дополнительной информации смотри следующий раздел и главу 3), то вы можете использовать команду

git pull. Она автоматически извлекает и затем сливает данные из удалённой ветки в вашу текущую ветку. Этот способ может для вас оказаться более простым или более удобным. К

тому же по умолчанию команда git clone автоматически настраивает вашу локальную

ветку master на отслеживание удалённой ветки master на сервере, с которого вы клонировали (подразумевается, что на удалённом сервере есть ветка master). Выполнение git pull, как правило, извлекает (fetch) данные с сервера, с которого вы изначально склонировали, и

автоматически пытается слить (merge) их с кодом, над которым вы в данный момент работаете.

1. Когда вы хотите поделиться своими наработками, вам необходимо отправить (push) их в главный репозиторий. Команда для этого действия простая:   
   git push [удал. сервер] [ветка].   
   Чтобы отправить вашу ветку master на сервер origin (повторимся, что клонирова ние, как правило, настраивает оба этих имени автоматически), вы можете выполнить следующую команду для отправки наработок на сервер:

$ git push origin master  
  
Эта команда срабатывает только в случае, если вы клонировали с сервера, на котором у вас есть права на запись, и если никто другой с тех пор не выполнял команду push. ИНАЧЕ Вам придётся сначала вытянуть (pull) их изменения и объединить с вашими. Только после этого вам будет позволено выполнить push.

1. Инспекция удалённого репозитория.

Если хотите получить побольше информации об одном из удалённых репозиториев,вы можете использовать команду git remote show [удал. сервер].

1. Удаление и переименование удалённых репозиториев.

Для переименования ссылок можно вылолнить git remote rename, это изменит сокращённое имя, используемое для удалённого репозитория. Например, если вы хотите переименовать pb в paul, вы можете сделать это следующим образом:

$ git remote rename pb paul

$ git remote

origin

paul

Стоит упомянуть, что это также меняет для вас имена удалённых веток. То, к чему выобращались как pb/master, стало paul/master.

Если по какой-то причине вы хотите удалить ссылку (вы сменили сервер или больше

не используете определённое зеркало, или, возможно, контрибьютор перестал быть активным), вы можете использовать git remote rm:

$ git remote rm paul

$ git remote

origin

1. Просмотр меток.

Просмотр имеющихся меток (tag) в Git’е делается просто. Достаточно набрать git tag:

$ git tag

v0.1

v1.3

Данная команда перечисляет метки в алфавитном порядке; порядок их появления не

имеет значения. Для меток вы также можете осуществлять поиск по шаблону. Например, репозиторий Git’а содержит более 240 меток. Если вас интересует просмотр только выпусков 1.4.2, вы можете выполнить следующее:

$ git tag -l 'v1.4.2.\*'

v1.4.2.1

v1.4.2.2

v1.4.2.3

v1.4.2.4

1. Создание аннотированной метки в Git’е выполняется легко. Самый простой способ это указать -a при выполнении команды tag:

$ git tag -a v1.4 -m 'my version 1.4'

$ git tag

v0.1

v1.3

v1.4

Опция -m задаёт меточное сообщение, которое будет храниться вместе с меткой. Если не указать сообщение для аннотированной метки, Git запустит редактор, чтоб вы смогли его ввести.  
Вы можете посмотреть данные метки вместе с коммитом, который был помечен, с помощью команды git show:

$ git show v1.4

1. Создадим новую ветку под названием testing. Это делается командой git branch:

$ git branch testing  
В данный момент вы всё ещё на ветке master. Команда git branch только создала новую ветку, она не переключила вас на неё.  
Чтобы перейти на существующую ветку, вам надо выполнить команду git checkout.

Давайте перейдём на новую ветку testing:

$ git checkout testing  
Откуда Git узнает, на какой ветке вы находитесь в данный момент? Он хранит специальный указатель, который называется HEAD (верхушка). В Git’е это указатель на локальную ветку, на которой вы находитесь.  
Чтобы создать ветку и сразу же перейти на неё, вы можете выполнить команду git checkout с ключом -b:

$ git checkout -b iss53

Switched to a new branch "iss53"

Это сокращение для:

$ git branch iss53

$ git checkout iss53

1. Вы можете удалить ветку с помощью опции -d к git branch:

$ git branch -d hotfix

Deleted branch hotfix (3a0874c).

1. Допустим, вы разобрались с проблемой №53 и готовы объединить эту ветку и свой master. Чтобы сделать это, мы сольём ветку iss53 в ветку master точно так же, как мы делали это ранее с веткой hotfix. Всё, что вам нужно сделать, — перейти на ту ветку, в которую вы хотите слить свои изменения, и выполнить команду git merge:

$ git checkout master

$ git merge iss53

Пункт Pravka3.  
Рисуем схему. Вставляем её перед синим текстом ветки мастер.

Прав2.Ком1

Ком1

Коммит1

Прав1.Ком2

Прав1.Ком1

Коммит2

1. (Редактируем пункт. Ветка pravka1. Изменяем цвет на синий).   
   Команда git branch делает несколько больше, чем просто создаёт и удаляет ветки. Если вы выполните её без аргументов, то получите простой список имеющихся у вас веток:

$ git branch

iss53

\* master

testing

Обратите внимание на символ \*, стоящий перед веткой master: он указывает на ветку, на которой вы находитесь в настоящий момент. Это означает, что если вы сейчас выполните

коммит, ветка master переместится вперёд в соответствии с вашими последними измене-

ниями. Чтобы посмотреть последний коммит на каждой из веток, выполните команду git

branch -v:

$ git branch -v

iss53 93b412c fix javascript issue

\* master 7a98805 Merge branch 'iss53'

testing 782fd34 add scott to the author list in the readmes  
  
ДЕЛАЕМ КОММИТ 1 НА ВЕТКЕ PRAVKA1  
Добавляем этот текст другим цветом.

ДЕЛАЕМ КОММИТ 2 НА ВЕТКЕ PRAVKA1