Git

Git

Git – свободная распределенная система управления версиями. Проект был создан Линусом Торвальдсом для управления разработкой ядра Linux (2005). Позже стал использоваться во многих других проектах – таких как, например, Android, Chromium и многих других.

С точки зрения реализации Git представляет собой набор утилит командной строки, работа которых может управляться параметрами. Вся информация хранится в обычных текстовых файлах.

Самая главная команда

Если вам нужна помощь при использовании Git, есть три способа открыть страницу руководства по любой команде Git:

- git help <глагол>
- git <глагол> --help
- man git-<глагол>

Настройки

В состав Git входит утилита git config, которая позволяет просматривать и настраивать параметры, контролирующие все аспекты работы Git.

Эти параметры могут быть сохранены на трех уровнях:

- на уровне ОС;
- на уровне пользователя;
- на уровне репозитория.

Настройки

- Настройки на каждом следующем уровне подменяют настройки из предыдущих уровней.
- Команда *git config --list* показывает настройки.
- Первое, что необходимо сделать, указать ваше имя и адрес электронной почты.

```
$ git config user.name "IgorL"
$ git config user.email ilomovskoy@bmstu.ru
```

- Если указана опция *--global*, то эти настройки достаточно сделать только один раз.

Создание репозитория

Для создания Git-репозитория можно использовать два основных подхода:

- импорт в Git уже существующего проекта или директории (*git import*);
- клонирование существующего репозитория с другого сервера (*git clone*).

Создание репозитория

Если вы собираетесь начать использовать Git для существующего проекта, то необходимо перейти в директорию проекта и в командной строке ввести

```
$ cd project_name
$ git init
Инициализирован пустой репозиторий Git в .../.git/
```

Структура репозитория

С точки зрения Git директория проекта делится на

- рабочую директорию (working directory), содержащую изменяемые файлы;
- индекс (index или staging area), содержащий информацию о том, какие изменения попадут в следующую фиксацию;
- репозиторий.

Структура репозитория

Репозиторий Git представляет собой каталог файловой системы (.git), в котором находятся:

- файлы конфигурации репозитория,
- файлы журналов, хранящие операции, выполняемые над репозиторием,
- индекс, описывающий расположение файлов,
- хранилище, содержащее собственно файлы.

Состояния файлов

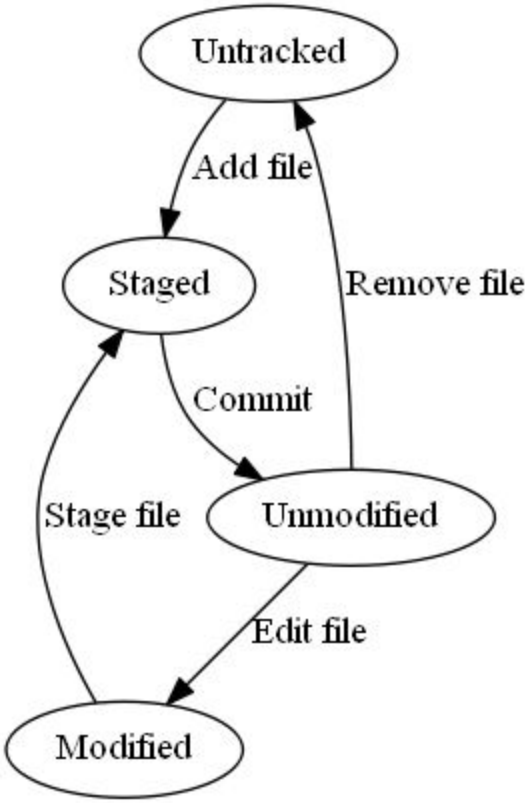
С точки зрения git файлы, находящиеся в рабочей копии, могут находиться в следующих состояниях:

- отслеживаемые (под версионным контролем);
- неотслеживаемые.

Отслеживаемые файлы, в свою очередь, могут находиться в следующих состояниях:

- зафиксированное (committed);
- измененное (modified);
- подготовленное (staged/cached).

Состояния файлов



Untracked	Неотслеживаемый
Staged	Подготовленный
Unmodified	Зафиксированный
Modified	Измененный
Add file	Планирование для включения в фиксацию
Stage file	
Edit file	Изменение файла
Remove file	Удаление из-под верс. контр.
Commit	Фиксация

Определение состояния файла

Основной инструмент, используемый для определения, какие файлы в каком состоянии находятся – команда *git status*.

```
$ git status
На ветке master

Начальный коммит

нечего коммитить (создайте/скопируйте файлы, затем запустите «git
   add», чтобы отслеживать их)
```

Определение состояния файла

Поместим файлы в рабочую директорию.

```
$ ls
iarray.py  main.py

$ git status
На ветке master

Начальный коммит

// секция “Untracked files”
Неотслеживаемые файлы:
  (используйте «git add <файл>…», чтобы добавить в то, что ...

        iarray.py
        main.py

ничего не добавлено в коммит, но есть неотслеживаемые файлы (...
```

Добавление файлов в репозиторий

Для того чтобы начать отслеживать (добавить под версионный контроль) новый файл, используется команда *git add*.

```
$ git add iarray.py main.py

$ git status
На ветке master

Начальный коммит

// секция “Changes to be committed”
Изменения, которые будут включены в коммит:
  (используйте «git rm --cached <файл>…», чтобы убрать из индекса)

        новый файл:    iarray.py
        новый файл:    main.py
```

Добавление файлов в репозиторий

Зафиксируем изменения в репозитории (команда *git commit*).

```
$ git commit -m "Initial version of program."
[master (корневой коммит) 88a2bc9] Initial version of program.
 2 files changed, 37 insertions(+)
 create mode 100644 iarray.py
 create mode 100644 main.py
```

Git для идентификации ревизий использует значение хэша (SHA-1) фиксации (88a2bc9...).

Исправление ошибки

```
def Test2():
 Arr = list()

 Arr.append(1)
 Arr.append(1)
 Arr.append(5)
 Arr.append(5)
 Arr.append(5)

 return Arr, 5

def main():
 ...
 Arr, N = Test2()
 print(...GetMaxCount(Arr, N)...
```

```
def GetMaxCount(Arr, N):
  Max = Arr[0]
  Count = 1

  I = 1
  while (I < N):
    if (Arr[I] > Max):
      Max = Arr[I]
      Count = 1
    else:
      if (Max == Arr[I]):
        Count += 1;

    I += 1

  return Count
```

Анализ изменений

Команда git status показывает какие файлы изменились, но не показывает, что в них изменилось.

```
$ git status
На ветке master
Изменения, которые не в индексе для коммита:
  (используйте «git add <файл>…», чтобы добавить файл в индекс)
  (используйте «git checkout -- <файл>…», чтобы отменить ...
        изменено:       iarray.py
        изменено:       main.py
Неотслеживаемые файлы:
  (используйте «git add <файл>…», чтобы добавить в то, что ...
        __pycache__/
нет изменений добавленных для коммита
(используйте «git add» и/или «git commit -a»)
```

Анализ изменений

Чтобы проанализировать добавленные и удаленные строки нужно воспользоваться командой *git diff*.

Команда *git diff* показывает изменения, которые еще не проиндексированы.

См. diff_1.txt

diff:универсальный формат

Команда git diff выводит разницу между двумя файлами.

Вывод команды представляет собой изменения, которые нужно произвести в исходном файле, чтобы получить новый файл. Для представления разницы используется так называемый универсальный формат unidiff (https://ru.wikipedia.org/wiki/Diff).

diff:универсальный формат

Имя исходного файла начинается с «---», а имя нового файла с «+++».

За именами следует один или больше измененных *фрагментов*, которые содержат построчные изменения в файлах. Строки без изменений начинаются с пробела, добавленные строки начинаются со знака плюс, удаленные строки начинаются со знака минус. Неизмененные строки (или *контекст*) служат ссылкой для определения положения изменяемого фрагмента в новом файле.

diff:универсальный формат

- Информация о диапазоне измененных строк (номер, количество) отмечена знаками @@.
- Информация о диапазоне состоит из двух частей. Часть для исходного файла начинается с минуса, а часть для нового файла начинается с плюса.
- Каждая часть в формате l, s, где l - номер строки, с которой начинаем, а s - количество строк, которые были изменены в текущем фрагменте для каждого из файлов, соответственно

Анализ изменений

Команда *git diff* показывает только те изменения, которые ещё не проиндексированы. Т.е. если вы *проиндексируете* все свои изменения, то *git diff ничего не вернёт* (!).

Если вы хотите посмотреть, какие изменения попадут в следующую фиксацию, необходимо выполнить команду *git diff --staged*. Эта команда сравнивает ваши индексированные изменения с репозиторием.

Анализ изменений

```
$ git add main.py  // проиндексируем изменения
$ git status
На ветке master
Изменения, которые будут включены в коммит:
  (используйте «git reset HEAD <файл>…», чтобы убрать из индекса)
        изменено:      main.py
Изменения, которые не в индексе для коммита:
  (используйте «git add <файл>…», чтобы добавить файл в индекс)
  (используйте «git checkout -- <файл>…», чтобы отменить ...
        изменено:      iarray.py
Неотслеживаемые файлы:
  (используйте «git add <файл>…», чтобы добавить в то, ...
        __pycache__/

$ git diff main.py  // команда ничего не вывела!
$ git diff --staged
```

См. diff_2.txt.

Анализ изменений

Зафиксируем main.py. Проиндексируем изменения в файле iarray.py, после чего внесем в файл iarray.py новые изменения.

```
$ git add iarray.py
// файл iarray.py был изменен
$ git status
На ветке master
Изменения, которые будут включены в коммит:
  (используйте «git reset HEAD <файл>…», чтобы убрать из индекса)
        изменено:      iarray.py  // !
Изменения, которые не в индексе для коммита:
  (используйте «git add <файл>…», чтобы добавить файл в индекс)
  (используйте «git checkout -- <файл>…», чтобы отменить ...
        изменено:      iarray.py  // !
Неотслеживаемые файлы:
  (используйте «git add <файл>…», чтобы добавить в то, что ...
        __pycache__/
```

Анализ изменений

Git индексирует файл в том состоянии, в котором он находился, когда выполнялась команда *git add*. Если выполнить фиксацию сейчас, то файл iarray.py попадет в репозиторий в том состоянии, в котором он находился, когда была выполнена команда *git add*, а не в том, в котором он находится в рабочей директории в момент выполнения *git commit*.

Если необходимо, чтобы изменения, внесенные после выполнения *git add*, попали в индекс, нужно еще раз выполнить команду *git add*.

Анализ изменений (задача)

```
$ git diff --staged
diff --git a/iarray.py
b/iarray.py
index bfe46b8..08acd73 100644
--- a/iarray.py
+++ b/iarray.py
@@ -6,6 +6,7 @@ def
GetMaxCount(Arr, N):
   while (I < N):
     if (Arr[I] > Max):
       Max = Arr[I]
+      Count = 1^M
     else:
       if (Max == Arr[I]):
         Count += 1;
```

```
$ git diff
diff --git a/iarray.py
b/iarray.py
index 08acd73..06312d3 100644
--- a/iarray.py
+++ b/iarray.py
@@ -9,7 +9,7 @@ def
GetMaxCount(Arr, N):
       Count = 1
     else:
       if (Max == Arr[I]):
-        Count += 1;
+        Count += 1^M
     I += 1
```

Откат изменений

Если вы случайно добавили файл в индекс и хотите отменить это добавление.

```
$ git status -s  // -s – краткая форма выдачи
   ?? __pycache__/
   ?? roll_back.txt
$ git add roll_back.txt
$ git status
   На ветке master
   Изменения, которые будут включены в коммит:
   (используйте «git reset HEAD <файл>...», чтобы убрать из индекса)
новый файл: roll_back.txt
...

   $ git reset HEAD roll_back.txt
$ git status -s
   ?? __pycache__/
   ?? roll_back.txt
```

Откат изменений

Вы внесли изменения в файл, который находится под версионным контролем, но поняли, что не хотите их сохранять.

```
$ git status
   На ветке master
   Изменения, которые не в индексе для коммита:
   (используйте «git add <файл>…», чтобы добавить файл в индекс)
   (используйте «git checkout -- <файл>…», чтобы отменить
   изменения
   в рабочем каталоге)
изменено: main.py
нет изменений добавленных для коммита
   (используйте «git add» и/или «git commit -a»)
$ git checkout -- main.py
$ git status -s
```

Откат изменений

Для того чтобы удалить файл из Git, вам необходимо удалить его из индекса с помощью команды *git rm*, а затем выполнить фиксацию. Команда *git rm* также удаляет файл из вашей рабочей директории.

Если вы хотите удалить фал из индекса, но при этом оставить его в рабочей директории необходимо использовать команду *git rm --cached* .

Анализ истории изменений

Для анализа истории изменений используется команда *git log*.

По умолчанию (т.е. без аргументов) *git log* перечисляет фиксации, сделанные в репозитории в обратном к хронологическому порядке (последние фиксации находятся вверху). Фиксации перечисляются с их SHA-1 хэшами, именем и электронной почтой автора, датой создания и сообщением фиксации.

Анализ истории изменений

-p	Показывает патч для каждого фиксации.
--name-status	Показывает список файлов, которые добавлены/изменены/удалены.
--graph	Отображает ASCII граф с ветвлениями и историей слияний.
--pretty	Показывает коммиты в альтернативном формате. Возможные варианты опций: oneline (--pretty=oneline), short, full, fuller и format (с помощью последней опции вы можете указать свой формат).

Анализ истории изменений

Выяснить, что делают команды:

git show commit_hash

git diff commit_hash_1 commit_hash_2

git diff commit_hash_1 commit_hash_2 file_name

Игнорирование файлов

Обычно существует группа файлов, которые вы не только не хотите добавлять в репозиторий, но и видеть в списках неотслеживаемых. К таким файлам обычно относятся автоматически генерируемые файлы (например, объектные и исполняемые файоы).

В таком случае, вы можете создать файл *.gitignore* с перечислением шаблонов соответствующих таким файлам.

Игнорирование файлов

*.exe	* соответствует 0 или более символам (в данном случае – все файлы с расширение exe)
*.[oa]	[ao] соответствует любому символу из указанных в скобках (в данном случае – все файлы с расширение o или a)
**	** указывают вложенные директории (например, a/**/z будут соответствовать a/b/z или a/b/c/z или a/b/c/d/z и т.д.)

Литература

https://git-scm.com/book/ru/v2 (главы 1, 2)