Почему GIT?

Независимо от выбранного языка или направления разработки, код, который пишет программист, остаётся обычным текстом, записанным в множестве файлов на диске. Эти файлы регулярно добавляются, удаляются и изменяются. Некоторые из них могут содержать сотни строчек кода, а другие тысячи. Файлы в тысячу строк кода, вполне нормальное явление в программировании.

Пока проект состоит из пары-тройки файлов, его разработка не создаёт никаких сложностей. Программист пишет код, запускает его и радуется жизни. Клиент доволен, заказчик тоже. С ростом кодовой базы, появляются определённые неудобства, которые затем превращаются в реальные проблемы:

- Как не потерять файлы с исходным кодом?
- Как защититься от случайных исправлений и удалений?
- Как отменить изменения, если они оказались некорректными?
- Как одновременно поддерживать рабочую версию и разработку новой?

Решением является контроль версий. Выполняется он с помощью специальных программ, которые умеют отслеживать изменения кода. Вот некоторые из многочисленных возможностей данных систем:

- Возврат к любой версии кода из прошлого
- Просмотр истории изменений
- Совместная работа без боязни потерять данные или затереть чужую работу

Git – распределенная система управления версиями (VCS). Это универсальный, свободный и удобный инструмент для командной работы программистов над проектами любого уровня. **Git** позволяет нескольким разработчикам работать одновременно над своими подзадачами, создавая равноправные ветви. При этом каждое сохранение (коммит) в **Git** не перезаписывает предыдущее, и в любой момент Вы сможете вернуться к исходной версии кода.

Как будет использоваться GIT на данном курсе?

В ходе данного курса, Вы будете использовать **Git** для отправки Ваших решений домашних заданий. (Код программы, без лишних локальных файлов, должен заливаться в Гитхаб или ГитЛаб в открытый репозиторий, решение задачи оформляется в пакете с номером задачи , например part1.lesson01.task01)

Подробнее об отправке Ваших решений в репозиторий (удаленное хранилище кода) читайте в правилах оформления ДЗ, либо уточняйте у Вашего наставника на первой встрече.

Где искать необходимую информацию по изучению GIT?

Полное руководство по Git:
https://git-scm.com/book/ru/v2

Быстрый старт:
https://githowto.com/ru
(главы 1 -8)

Самая короткая инструкция на русском языке:
http://rogerdudler.github.io/git-guide/index.ru.html

Визуальная справка по командам:
http://marklodato.github.io/visual-git-guide/index-ru.html

Полноценный курс по Git:
https://ru.hexlet.io/courses/intro_to_git

Шпаргалка по командам:
https://github.github.com/training-kit/downloads/ru/github-git-cheat-sheet/

Устанавливается Git как и любая другая программа:

- MacOS: brew install git (чтобы устанавливать через brew, нужно сначала установить homebrew)
- Ubuntu/Debian: sudo apt install git
- Windows: choco install git (Chocolatey либо Git for Windows, если вы не знакомы с пакетными менеджерами)

(Более подробно по ссылке.)

Проверяем, что Git установился:
`$ git --version`

Первым делом, необходимо выполнить настройку Git. С помощью команд указанных ниже, установите имя и почту.
`$ git config --global user.name "Mona Lisa"`
`$ git config --global user.email "mona@lisa.io"`

Эти данные используются в коммитах для понимания того, кто был автором коммита.

Инициализация репозитория

Для инициализации нужно выполнить команду git init внутри той директории, которая станет репозиторием.

```
~$ mkdir example
~$ cd example/
# Команда инициализации
example$ git init
Initialized empty Git repository in .../example/.git/
```

Инициализация создаёт директорию .git.

```
example$ ls -la
total 0
drwxr-xr-x   3 mokevnin  wheel   96 Mar  4 08:51 .
drwxrwxrwt  11 root      wheel  352 Mar  4 08:50 ..
drwxr-xr-x   9 mokevnin  wheel  288 Mar  4 08:51 .git
```

Внутри неё содержится конфигурация репозитория (в файле .git/config) и все изменения, которые когда-либо делались с репозиторием. Её удаление равносильно удалению репозитория. При этом код, с которым мы непосредственно работаем (то, что лежит вне .git) находится в так называемой рабочей копии. Рабочая копия называется копией не случайно. Удалив все её содержимое, мы ничего не теряем. Git всё помнит и позволяет восстановиться до исходного состояния.

Убедиться в том, что репозиторий инициализирован и находится в работоспособном состоянии можно командой git status:

```
example$ git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)
```

Если выполнить команду git status вне директории содержащей .git, то Git «пожалуется» на отсутствие репозитория:

```
tmp$ git status
fatal: Not a git repository (or any of the parent directories): .git
```

Коммит

```
# Создаём файл
example$ touch README.md
# Меняем содержимое
example$ echo '# Hi' > README.md
# Так Git увидит новый файл
example$ git add README.md
 # Коммит с сообщением 'init project'
example$ git commit -m 'init project'
[master (root-commit) 679e31d] init project
 1 file changed, 1 insertion(+)
 create mode 100644 README.md
```

В коде выше две новых команды. Первая git add подготавливает изменённый или добавленный файл к коммиту. Без её выполнения сделать коммит не получится. А вот команда git commit непосредственно фиксирует изменения в репозитории. Ключ -m позволяет добавить короткое описание коммита. Если выполнить коммит без этого ключа, то откроется редактор с информацией о коммите. Git ожидает, что вы оставите описание коммита и выйдете из редактора. Дальше Git самостоятельно завершит свою работу.
Коммит в Git невозможно выполнить, не добавив к нему описание. Благодаря описанию мы можем просматривать историю коммитов и анализировать её без необходимости изучать изменения произведённые коммитом. Поэтому хорошее описание коммита очень важно. На просторах сети можно найти множество статей, посвящённых тому, как их именовать.

Рабочий процесс

Дальнейший рабочий процесс выглядит так:

1. Добавляем или изменяем файлы и директории.
2. Подготавливаем новое содержимое к коммиту командой git add. Эту команду нужно применять даже в том случае, если файл был просто изменён.
3. Выполняем коммит.

```
example$ echo 'my first change' >> README.md # Добавляем новую строку в файл
example$ git add README.md # Подготовка к коммиту
example$ git commit -m 'update readme'
[master 3a64fcc] update readme
 1 file changed, 1 insertion(+)

example$ cat README.md
# Hi
my first change
```

История: git log

Теперь можно посмотреть историю коммитов:

```
example$ git log

commit 3a64fccf14725593b7486ff09d6a6c325a5f8fcc
Author: Kirill Mokevnin <mokevnin@gmail.com>
Date:   Mon Jun 26 15:01:22 2017 +0300

    update readme

commit 679e31d62ac734e3074f092e417ba741be767a0d
Author: Kirill Mokevnin <mokevnin@gmail.com>
Date:   Mon Jun 26 14:35:47 2017 +0300

    init project
```

В выводе лога каждый коммит описывается таким набором данных:

- Идентификатор коммита, например, 679e31d62ac734e3074f092e417ba741be767a0d.
- Информация об авторе. Имя и адрес электронной почты, которые вводились во время конфигурации Git.
- Дата коммита.
- Описание коммита. Сообщение, которое вводилось при выполнении коммита.

Идентификатор коммита играет очень важную роль в жизненном цикле репозитория, и мы не раз к нему ещё вернёмся. Главное, что сейчас можно отметить: идентификатор представляет из себя хеш, а не последовательный номер (как в некоторых других системах).
Команда git log – очень мощный инструмент, у неё огромное количество опций, которые способны извлекать из истории все, что угодно в совершенно разных представлениях. Как правило, для нормальной повседневной работы, достаточно буквально нескольких разновидностей запросов к истории. Остальное легко ищется в интернете по мере надобности. Очень полезен такой вариант использования:

```
example$ git log -p

commit 3a64fccf14725593b7486ff09d6a6c325a5f8fcc
Author: Kirill Mokevnin <mokevnin@gmail.com>
Date:   Mon Jun 26 15:01:22 2017 +0300

    update readme

diff --git a/README.md b/README.md
index 76f177f..13f5a93 100644
--- a/README.md
+++ b/README.md
@@ -1 +1,2 @@
 # Hi
```

```
+my first change

commit 679e31d62ac734e3074f092e417ba741be767a0d
Author: Kirill Mokevnin <mokevnin@gmail.com>
Date:   Mon Jun 26 14:35:47 2017 +0300

    init project

diff --git a/README.md b/README.md
new file mode 100644
index 0000000..76f177f
--- /dev/null
+++ b/README.md
@@ -0,0 +1 @@
+# Hi
```

Он выводит непрерывную разницу (diff) всех изменений по коммитам. Строчки, в которых слева находится знак +, добавлены в этом коммите, а строчки, в которых есть - — удалены. Нажимая f и b в этом выводе, можно перемещаться по истории изменений если она не помещается на экран.

Просмотр изменений

Зная идентификатор коммита можно посмотреть изменения в коде.

```
$ git show 3a64fccf14725593b7486ff09d6a6c325a5f8fcc

diff --git a/README.md b/README.md
index 76f177f..13f5a93 100644
--- a/README.md
+++ b/README.md
@@ -1 +1,2 @@
 # Hi
+my first change
```

Вывод выше можно было бы получить и так:

```
$ git show 3a64fcc
```

То есть достаточно указать только первые 7 символов идентификатора коммита. Это сделано исключительно для удобства, так как вероятность коллизий (пересечений) крайне низка.
С другой стороны, команда git diff, запущенная без аргументов, показывает различия между репозиторием и вашей рабочей копией. Её полезно запускать тогда, когда вы хотите посмотреть изменения сделанные в рабочей копии, но не добавленные для коммита командой git add.
Сам вывод с непривычки кажется странным и единственный способ разобраться с ним – практика и только практика.

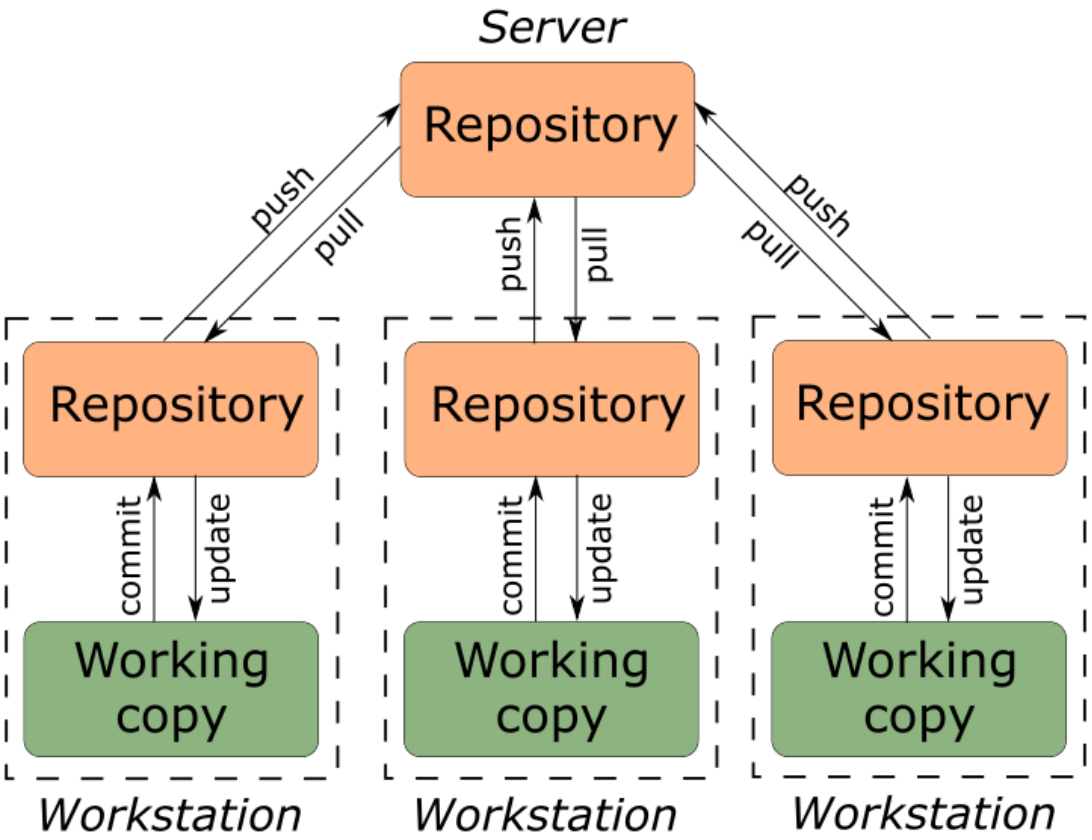
Самостоятельная работа

- На этом этапе можно начинать проходить https://git-scm.com/book/ru/v2.
- Установите Git. Убедитесь, что он работает: git --version.
- Сконфигурируйте Git, задав имя и почту.
- Создайте репозиторий и добавьте в него файл README.md. Измените файл и зафиксируйте изменения. Посмотрите историю изменений.

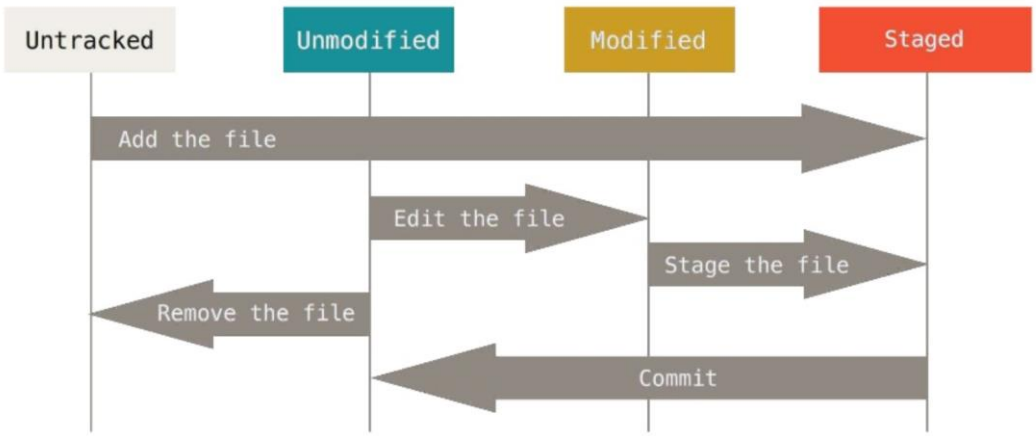
Что делать, если возникли вопросы/ проблемы по использованию GIT?

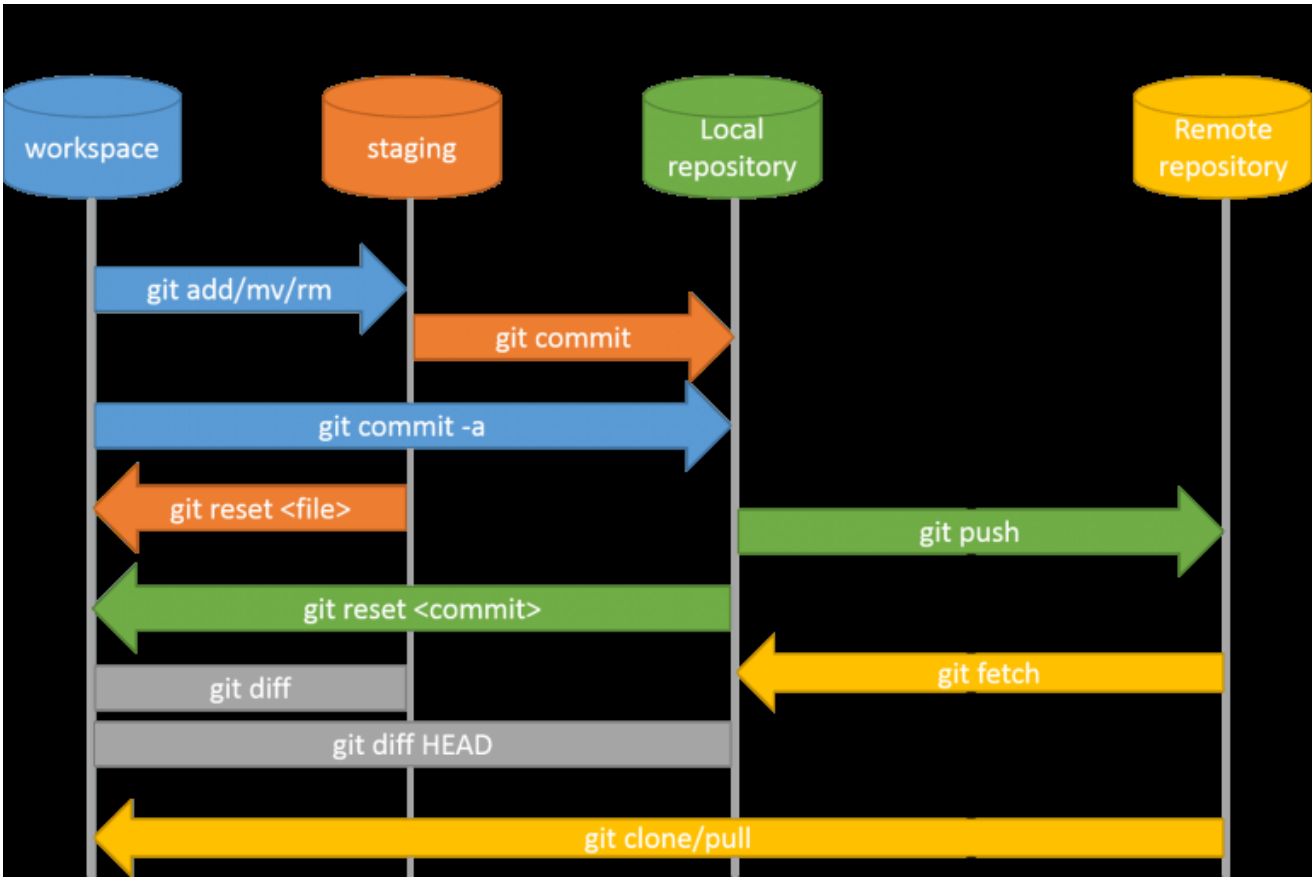
1. Проверить предложенные ссылки
2. На первой консультации наставники смогут ответить на возникающие вопросы по работе с Git, показать как он работает.

GIT в схемах



Git: files status lifecycle




workspace
staging
Local repository
Remote repository
git add/mv/rm
git commit
git commit -a
git reset <file>
git push
git reset <commit>
git fetch
git diff
git diff HEAD
git clone/pull