

Git Basics

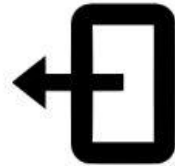
In case of fire



1. `git commit`



2. `git push`



3. leave building

Что такое Система контроля версий?

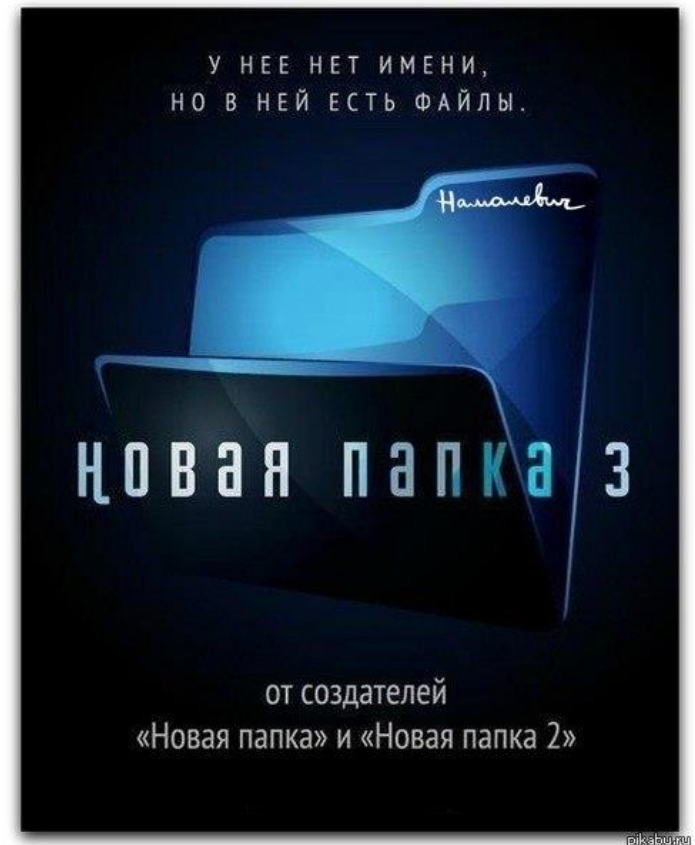
Система контроля версий — это система, регистрирующая изменения в одном или нескольких файлах для того, чтобы в дальнейшем была возможность вернуться к определённым старым версиям этих файлов.

- **Локальные**
- **Централизованные**
- **Распределенные**

Локальная система контроля версий

Этот вариант подразумевает простое копирование файлов в другой каталог, добавляя идентификатор версии к названию каталога. Это очень простой способ, но он в свою очередь чаще всего дает сбой.

Для решения этой проблемы была разработана **ЛСКВ** с простой базой данных, в которой хранятся все изменения нужных файлов.



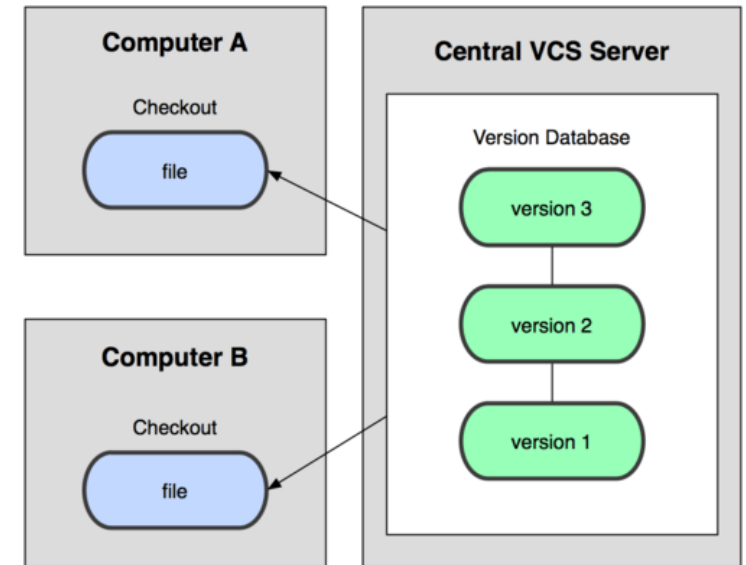
Централизованная система контроля версий

Следующей важной задачей оказалась необходимость координировать свои действия с разработчиками за другими компьютерами.

ЦСКВ представляет собой центральный сервер, на котором хранятся все файлы под версионным контролем, и ряд клиентов, которые из него получают копии файлов.

Преимущества: четкий контроль доступа и четкое понимание того, кто и какие изменения вносил в проект, и в какой момент времени.

Недостатки: централизованный сервер является самым уязвимым местом всей системы, т.к. вся история проекта хранится в одном месте.



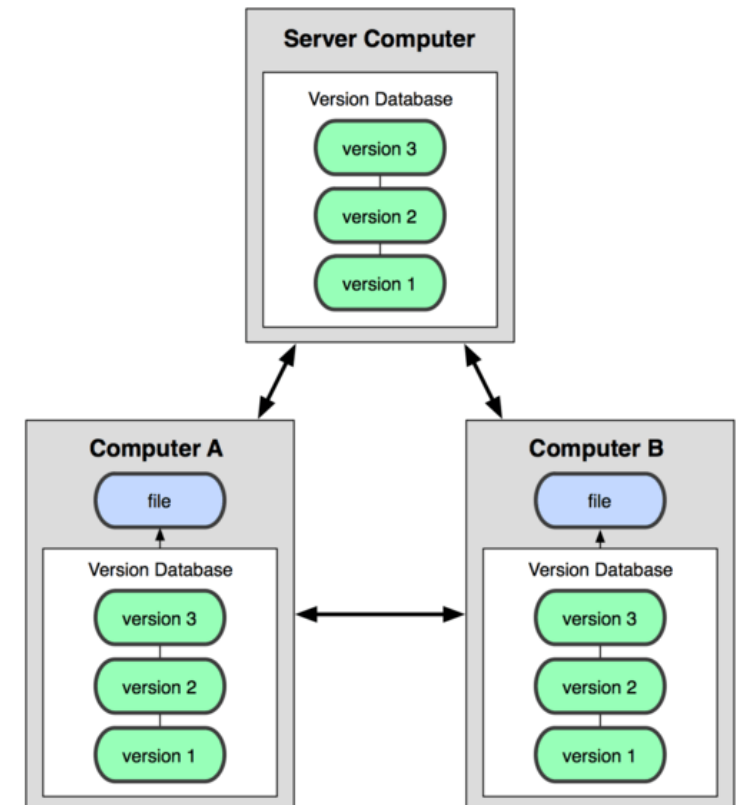
Распределенная система контроля версий

В таких системах клиенты не просто получают последние версии файлов, а **полностью копируют весь рабочий репозиторий** на свой рабочий компьютер.

Каждый раз, когда клиент забирает свежую версию файлов, он создаёт **себе полную копию всех данных**.

Если с сервером что-то случится, и данные по проекту будут потеряны, любой клиентский репозиторий может быть полностью **скопирован обратно на сервер для восстановления базы данных**.

В таких системах можно работать с несколькими удалёнными репозиториями, а это значит, что в рамках одного проекта **можно одновременно вести несколько рабочих процессов**.



Что такое Git?

Git — это **распределенная система контроля версий** для отслеживания изменений в файлах и координации работы над этими файлами среди большой команды людей.

Git. Альтернативы

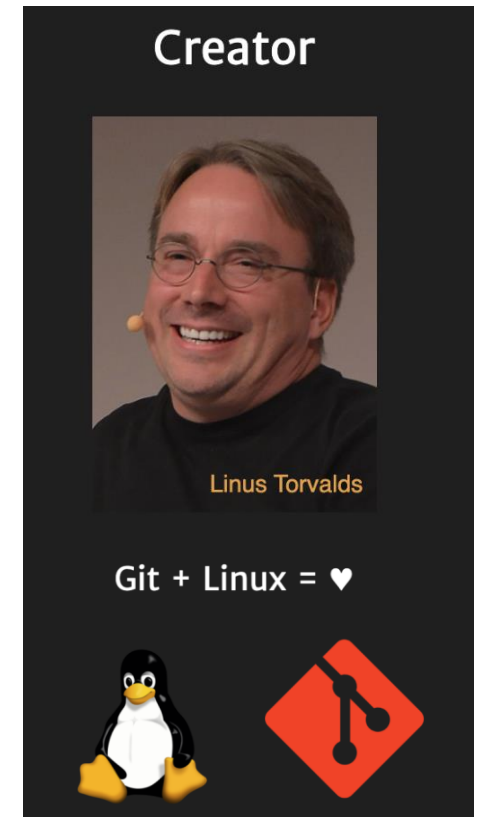
- **VS Team Foundation Server**
(Azure DevOps Server) – **РСКВ**
- **Mercurial** – **РСКВ**
- **Subversion** – **ЦСКВ**

Откуда взялся термин Git?

Git был создан Л. Торвальдсом в 2005 году для разработки ядра **Linux**, а другие разработчики ядра внесли свой вклад в его первоначальное развитие.

Имя **«git»** было дано Л. Торвальдсом, которое описывало этот инструмент как **«the stupid content tracker»**, а название:

- **random three-letter combination** that is pronounceable, and not actually used by any common UNIX command.
- **"global information tracker"**: you're in a good mood, and it actually works for you.
- **"g*dd*mn idiotic truckload of sh*t"**: when it breaks.



Популярные заблуждения о Git

Миф 1: Мне не нужен Git, потому что я делаю резервные копии своих файлов.

Миф 2: Git слишком сложный, чтобы заморачиваться.

Миф 3: Git только для команд разработчиков.

Миф 4: Командная строка это слишком сложно.

Миф 5: Я боюсь, что я что-нибудь сломаю.

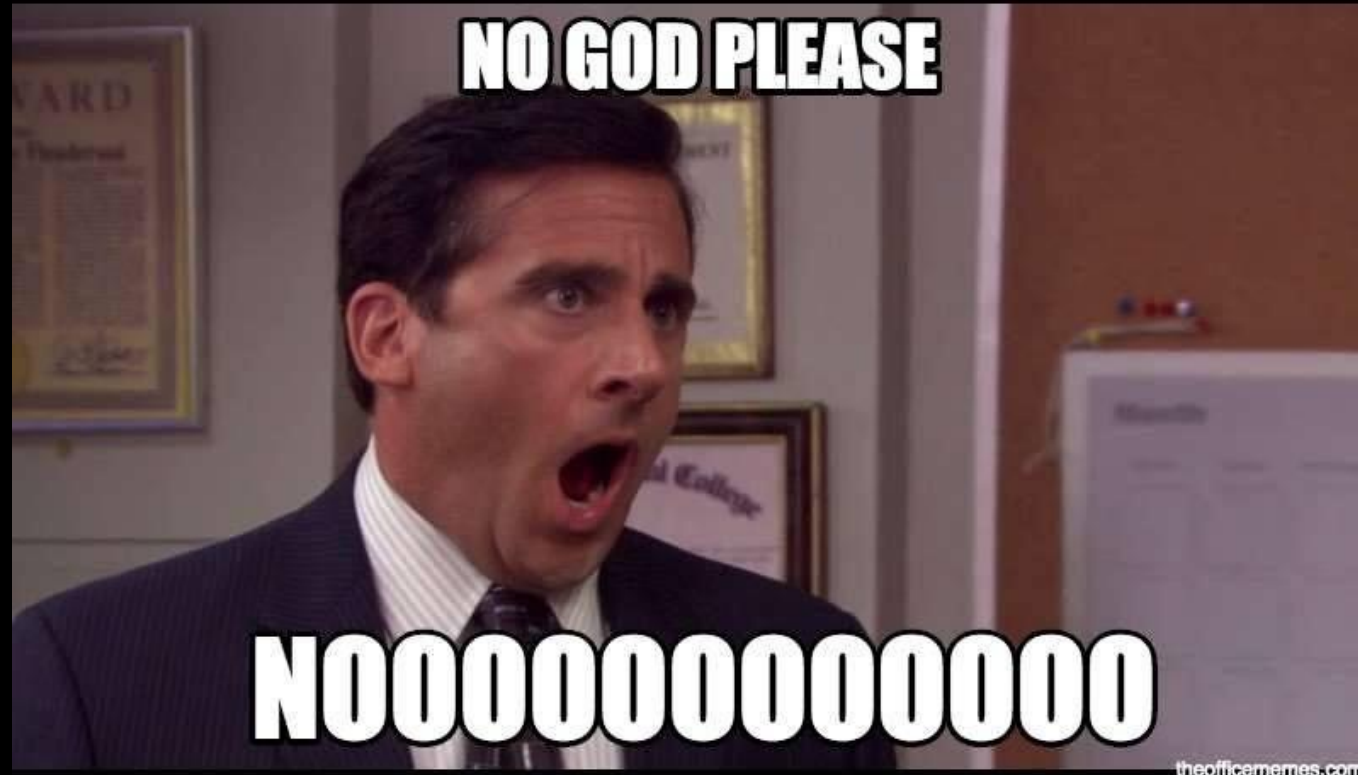
Миф 6: Git чрезмерно разрекламирован, это просто очередной модный тренд, который потом исчезнет.

Git. Почему это так важно?

Использование **системы контроля версий** позволяет Вам:

- **Хранить все изменения** на всех этапах разработки приложения (можно отследить все изменения в проекте **за любой промежуток времени!** и восстановить их в случае необходимости);
- **Разобраться**, как работал и работает нужный функционал; **найти** в какой момент времени он вышел из строя; а также **определить**, чьи изменения на это повлияли.
- **Release** и **Backup** (можно **хранить** релизную/финальную версию или, при необходимости, **откатиться** до любой стабильной версии продукта);
- **Распределить работу** между членами команды (каждый человек работает над своим функционалом приложения **независимо!**) и **собрать** все компоненты в единый проект.

GIT != GITHUB



**GitHub is a web-based hosting service for Git repositories (repos).
Think of it as a social network for creating software.**

Git. Терминология

- **Репозиторий** — папка с файлами и настройками **Git**. Настройки расположены в скрытой папке **.git**.
- **Fork** — копия репозитория. В таком случае выбранный вами сервис создаст копию фокнутого репозитория на вашем аккаунте. После этого в копию репозитория можно будет вносить изменения. Внесенные изменения можно предложить добавить в основной репозиторий через **pull request**.
- **Clone** — скачать репозиторий с удалённого сервера на локальный компьютер.
- **Ветка/Branch** — это параллельная версия репозитория. Ветки можно объединять между собой (например, соединять параллельную версию с основной версией репозитория).

Git. Терминология

- **Master** — главная и/или основная ветка репозитория.
- **Commit** — фиксация изменений или запись изменений в репозиторий. Каждый коммит происходит на локальном компьютере.
- **Pull** — получение последних изменений с удалённого сервера репозитория.
- **Push** — отправка всех неотправленных коммитов на удалённый сервер репозитория.
- **Merge** — слияние изменений из какой-либо ветки репозитория с любой веткой этого же репозитория.

Git. Терминология

- **Pull request** — запрос на слияние форка репозитория с основным репозиторием. **Pull request** может быть принят или отклонён владельцем репозитория.
- **Code review** — процесс проверки кода на соответствие определённым требованиям, задачам и внешнему виду (**Code style**).
- **Обновиться из upstream** — обновить свою локальную версию форка до последней версии основного репозитория, от которого сделан форк.
- **Обновиться из origin** — обновить свою локальную версию репозитория до последней удалённой версии этого репозитория.

С чего начать работу с Git?

1. Установите **Git** на Ваш компьютер:
 - Владельцы **OS Windows** – Вам необходимо скачать и установить **Git** по ссылке: <https://git-scm.com/download/win> .
 - Владельцы **Unix-систем**, таких как **OS Linux** и **MacOS X** – у Вас уже все предустановлено: **Git** доступен в **Terminal** по умолчанию.

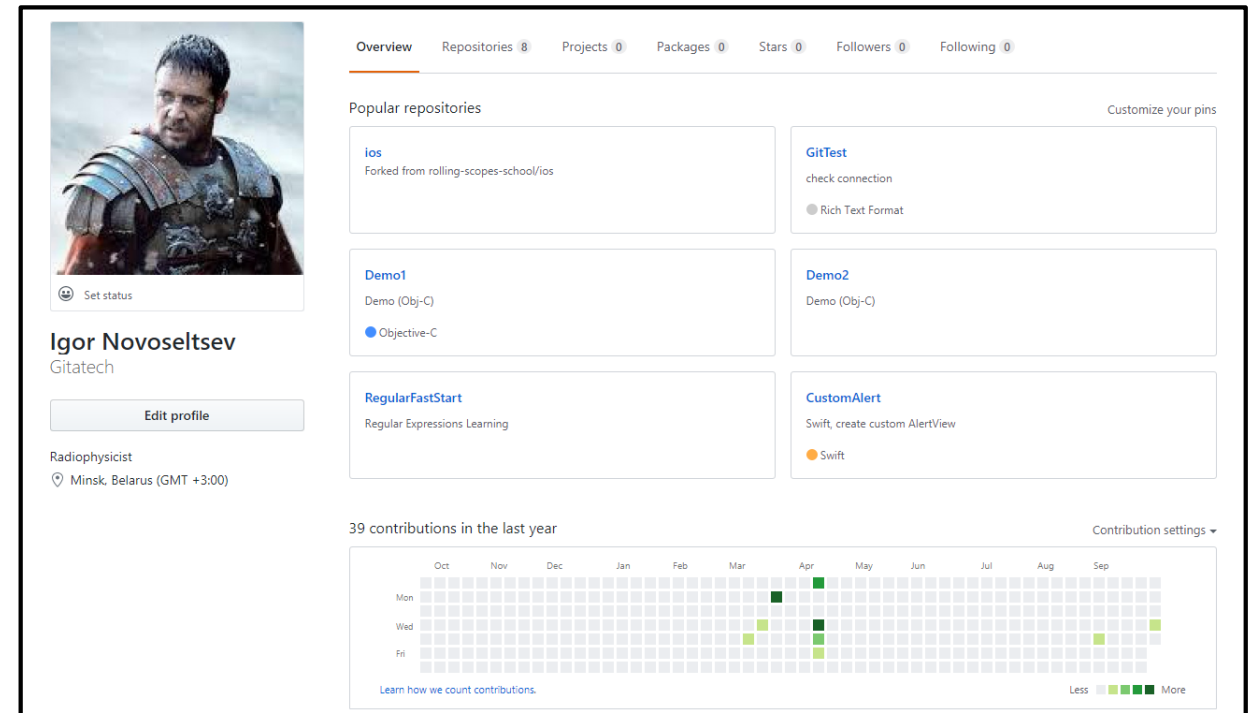


С чего начать работу с Git?

2. Создайте свой профиль на **GitHub/Gitlab/Bitbucket**.

В нем укажите ваши:


- ФИО
- Факультет
- Курс
- Специальность
- Группу



С чего начать работу с Git?

3. Создайте на **GitHub/Gitlab/Bitbucket** свой репозиторий:

Owner

 Gitatech ▾

/


Repository name *

myFirstRepository ✓


Great repository names are short and memorable. Need inspiration? How about [fuzzy-octo-guide?](#)

Description (optional)

Git Basics Education

☒  **Public**

Anyone can see this repository. You choose who can commit.

☐  **Private**


You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

☐ **Initialize this repository with a README**

This will let you immediately clone the repository to your computer.

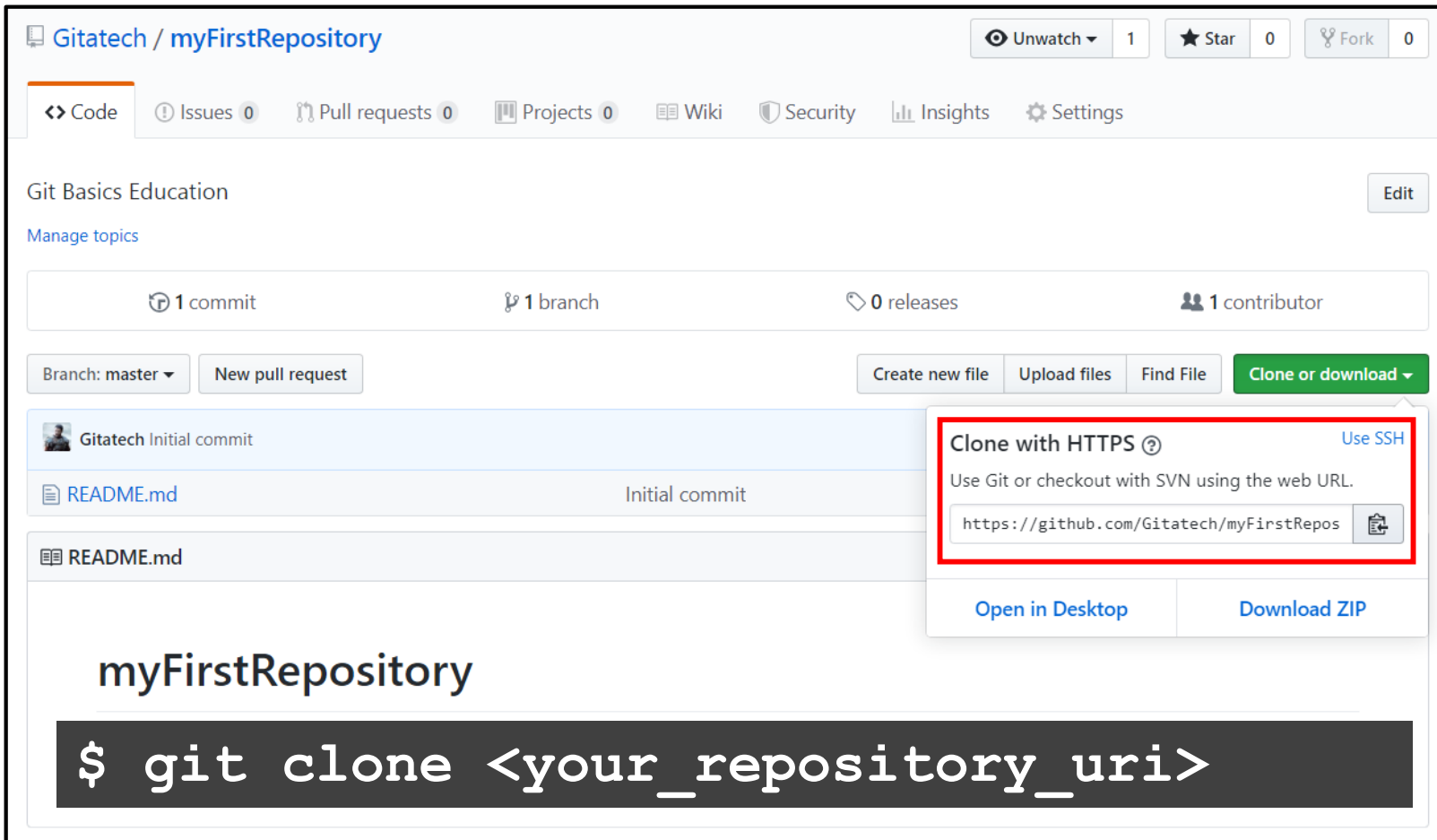
Add .gitignore: None ▾

Add a license: None ▾ 

Create repository

С чего начать работу с Git?

4. Загрузите Ваш репозиторий на компьютер через **HTTPS** или через **SSH**:



The screenshot shows the GitHub interface for a repository named 'Gitatech / myFirstRepository'. The repository has 1 commit, 1 branch, 0 releases, and 1 contributor. The 'Clone or download' button is highlighted, and a dropdown menu is open, showing the 'Clone with HTTPS' option selected. The dropdown menu also includes a 'Use SSH' link, a text field with the URL 'https://github.com/Gitatech/myFirstRepos', and a 'Copy' icon. Below the dropdown menu, there are buttons for 'Open in Desktop' and 'Download ZIP'. At the bottom of the page, there is a terminal window showing the command to clone the repository.

```
$ git clone <your_repository_uri>
```

Git. Настройка конфигурации

Перед работой с **Git** нужно настроить **имя** профиля и ваш **email**.

1. Настройка имени.

```
$ git config --global user.name "YourName"
```

- **git** — все команды для Git начинаются с этого слова.
- **config** — конфигурация.
- **--global** — означает, что настройка будет применена для всех проектов.
- **user.name** — настройка, которую нужно установить.
- **YourName** — имя, которое нужно установить.

Git. Настройка конфигурации

Перед работой с **Git** нужно настроить **имя** профиля и ваш **email**.

2. Настройка электронной почты:

```
$ git config --global user.email name@gmail.com
```

3. Проверяем, что настройки сохранились:

```
$ git config --global --list
```

** (--global, если настройки имени задавались для всех проектов и нужно вывести именно их)*

Среди прочих настроек, если они есть, Вы должны увидеть:

- **user.name=YourName**
- **user.email=name@gmail.com**

Git. Настройка конфигурации

```
$ git config --global --list  
    user.name=Igor Novoseltsev  
    user.email=inovoseltsev@outlook.com
```

```
$ git config --global user.name "John Doe"  
$ git config --global user.email johndoe@example.com
```

Git. Настройка конфигурации

Как открыть файл с настройками?

Настройки хранятся в файле **.gitconfig**:

```
$ cat ~/.gitconfig
```

(такого файла может не быть, если изначально не прописать настройки)

Если настройки успешно сохранились, там будет информация:

```
[user]
```

```
name = YourName
```

```
email = name@gmail.com
```

в том числе и другие настройки, если они есть

Ок, Вы потерялись в консоли, что делать?

Главное, не паникуйте. Вас точно спасет вот эта команда:

\$ git --help или **\$ git help ...**

*(справка по всем популярным командам **Git**)*

А также Вам помогут:

\$ git help <название_команды>

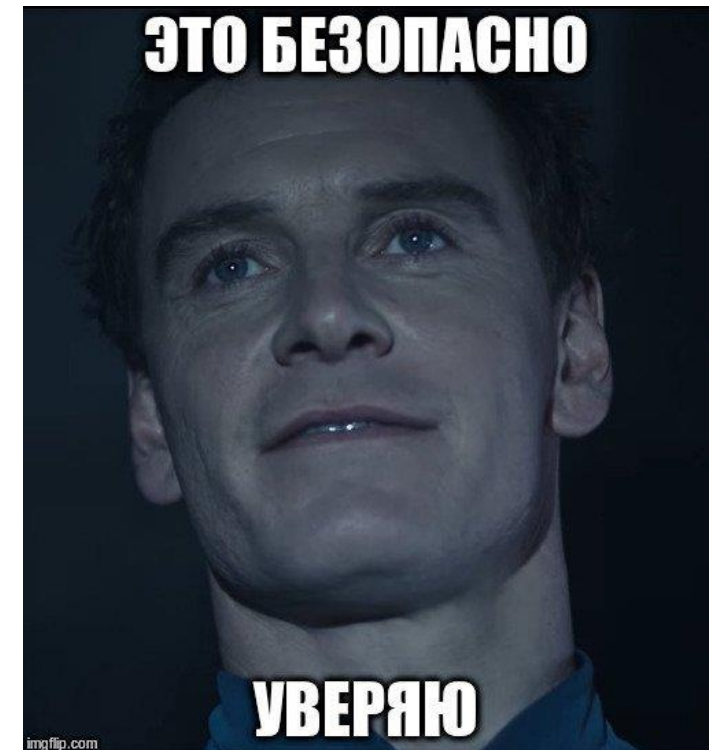
(справка по выбранной команде)

Например,

\$ git help config

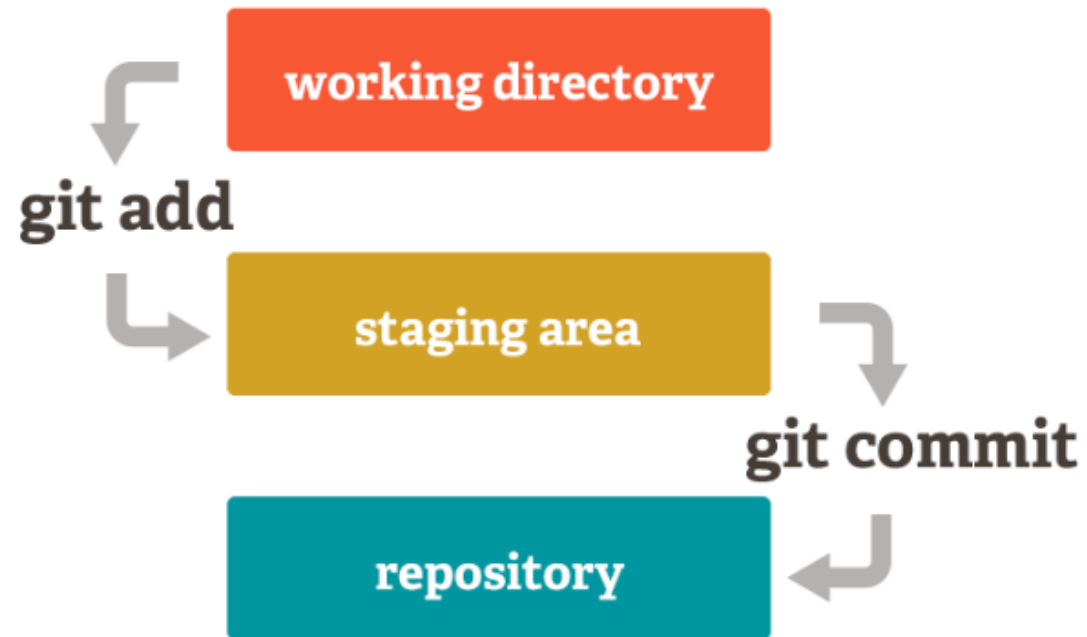
*(показывает справку для команды **config**)*

*Подробнее с возможностями **Helper**'а рекомендую ознакомиться самостоятельно.*



Git. Workflow

Workflow



Git. Workflow

1. Подготовка репозитория к работе:

Первое, что необходимо сделать, если Вы создали репозиторий локально или, если он ещё не привязан к Git:

```
$ git init
```

После этого в папке с проектом появится скрытая папка **".git"**, в которой будет храниться вся история изменений.

Git. Workflow

2. Посмотреть состояние репозитория:

Чтобы проверить, были ли добавлены новые изменения, которые ещё не внесены в историю изменений, можно использовать команду:

```
$ git status
```

Обратите внимание!

- Команда сработает, если напротив директории, в которой Вы находитесь, есть надпись **(master)**.
- Если такой надписи **нет**:
 - Вы находитесь в другой директории;
 - Ваша директория не содержит скрытую папку **.git**.

Git. Workflow

Как работает команда **\$ git status**?

- Если выводятся файлы, выделенные **красным**:
новые изменения не добавлены в Git (файлы не отслеживаются).
- Если файлы выводятся **зеленым**:
измененные файлы добавлены в Git (все файлы отслеживаются).
- Если нет подсвеченных файлов, а в консоли выведена надпись **«nothing to commit, working tree clean»**:
нет новых невнесенных изменений (все файлы отслеживаются).

Git. Workflow

3. Проиндексировать новые или измененные файлы:

- `$ git add .` (проиндексировать все файлы в папке).
- `$ git add index.html` (проиндексировать только файл index.html)
- `$ git add css/button.css css/main.css`
(проиндексировать сразу 2 файла, которые находятся в папке "css": **button.css** и **main.css**, указанные через пробел).

После рекомендуется вызвать команду `git status` — так можно проверить, все ли файлы были проиндексированы.

Git. Workflow

4. Зафиксировать изменения — закоммитить.

Позволяет сделать новые изменения отслеживаемыми. Таким образом всегда **можно откатиться к предыдущему закоммиченному изменению**.

Другими словами: **коммит** — аналогичен команде **save / checkpoint** в играх.

```
$ git commit -m "Описание того, что изменилось"
```

После вызова команды **git status** Вы сможете проверить, закоммитились ли файлы. Если все в порядке, то в консоль выведется сообщение: **"nothing to commit, working tree clear"**.

!!! Коммитить нужно всегда, когда состояние вашей части работы **обрело законченный вид**: добавили новый блок кода, исправили ошибку, внесли значимые изменения, потеря которых будет особо критична и т.п.

Git. Workflow

Иногда в описании коммитов **допускаются ошибки**.

В таком случае, Вам следует самостоятельно исправить последний коммит.

```
$ git commit --amend -m "Правильное сообщение коммита"
```

```
$ git commit -a -m "Правильное сообщение коммита"
```

Обратите внимание!

*После этого последний коммит с неправильным текстом **исчезнет из логов** и **hash последнего коммита изменится**.*

Получается, что это не старый коммит изменяется, а создается новый коммит вместо старого.

Git. Workflow

5. Как посмотреть, что изменилось?

Посмотреть все **непроиндексированные** изменения:

- `$ git diff` (Git выведет все, что изменилось в файле: **красным** — удаленное, **зеленым** — добавленное)

Посмотреть все **проиндексированные** изменения:

- `$ git diff --staged`

Git. Workflow

6. Как посмотреть историю коммитов?

- `$ git log` (покажет всю историю коммитов этого репозитория, если она есть)
- `$ git log -2` (покажет только 2 последних коммита этого репозитория)

В итоге, в консоль выведется следующая информация:

`c73af11e0830fe25cv4648c2356742f253cc000d` — hash второго коммита.

Зная hash коммита, можно заглянуть в лог коммита или откатиться к этому коммиту.

```
commit c73af11e0830fe25cv4648c2356742f253cc000d (HEAD -> master, origin/master)
```

```
Author: YourName <name@gmail.com>
```

```
Date: Thu Oct 03 13:01:09 2019 +0300
```

```
second commit
```

```
commit e5cd17321di6love8the2rs1school687dde6282
```

```
Author: YourName <name@gmail.com>
```

```
Date: Thu Oct 03 12:26:17 2019 +0300
```

```
first commit
```

Git. Workflow

Также можно вывести историю всех коммитов в одну строку.

```
$ git log --oneline
```

В консоль информация выведется в следующем виде:

```
5c71719 (HEAD -> master) Уменьшили шрифт на кнопках
b69e1e8 Уменьшили шрифт телефона
2c8bd55 Заменяли span на button в кнопках для лучшей доступности
6bf8b41 Сделали рефакторинг
0c5e111 Починили ошибку фильтрации
5949091 Изменили основной шрифт
5d33d23 Установили цвет кнопок и фона
4f485ba initial commit
```

Git. Workflow

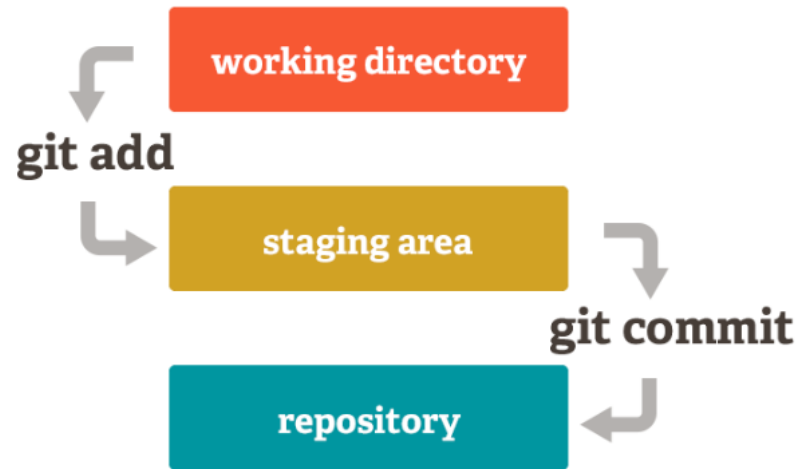
7. Как узнать, какие изменения были внесены в определенном коммите:

`$ git show hash-commit`

```
$ git show e5mnd6f07fb3n72f9ccc3b24839687d61cde6282
```

Git. Real world workflow

Workflow



Real world workflow

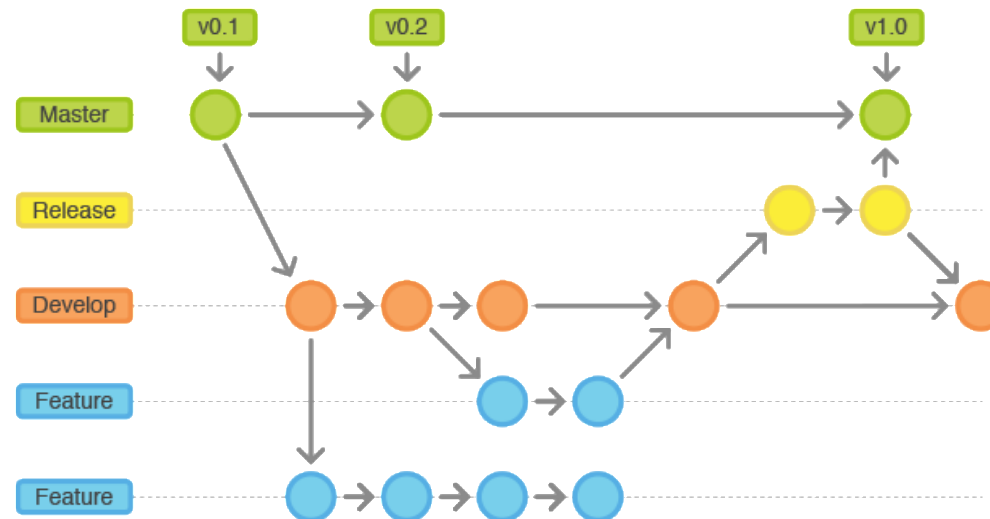


Git. Branches

Каждая **ветка** — это своё **собственное развитие исходного кода**.

Веток в проекте может быть сколько угодно, и порождаться они могут из любого коммита любой ветки (дерево же).

Они помогают разным людям работать параллельно над одним проектом — каждый в своей ветке. А затем эти ветки соединяются воедино — в один проект.



Git. Branches

Давайте разберемся как посмотреть все ветки, которые есть в проекте:

\$ git branch

Тогда в консоли выведутся все ветки Вашего проекта, где * - это текущая рабочая ветка, в которой вы находитесь.

```
$ git branch
```

```
feature/DIS-5708-checkouts
```

```
feature/DIS-7149-update-navigation
```

```
* master
```

Git. Branches

Для создания новой ветки Вам нужна команда:

```
$ git branch feature-branch
```

А чтобы переключиться на новую ветку, Вам понадобится команда:

```
$ git checkout feature-branch
```

```
$ git branch feature-branch
```

```
$ git branch
```

```
feature-branch
```

```
feature/DIS-5708-checkouts
```

```
feature/DIS-7149-update-navigation
```

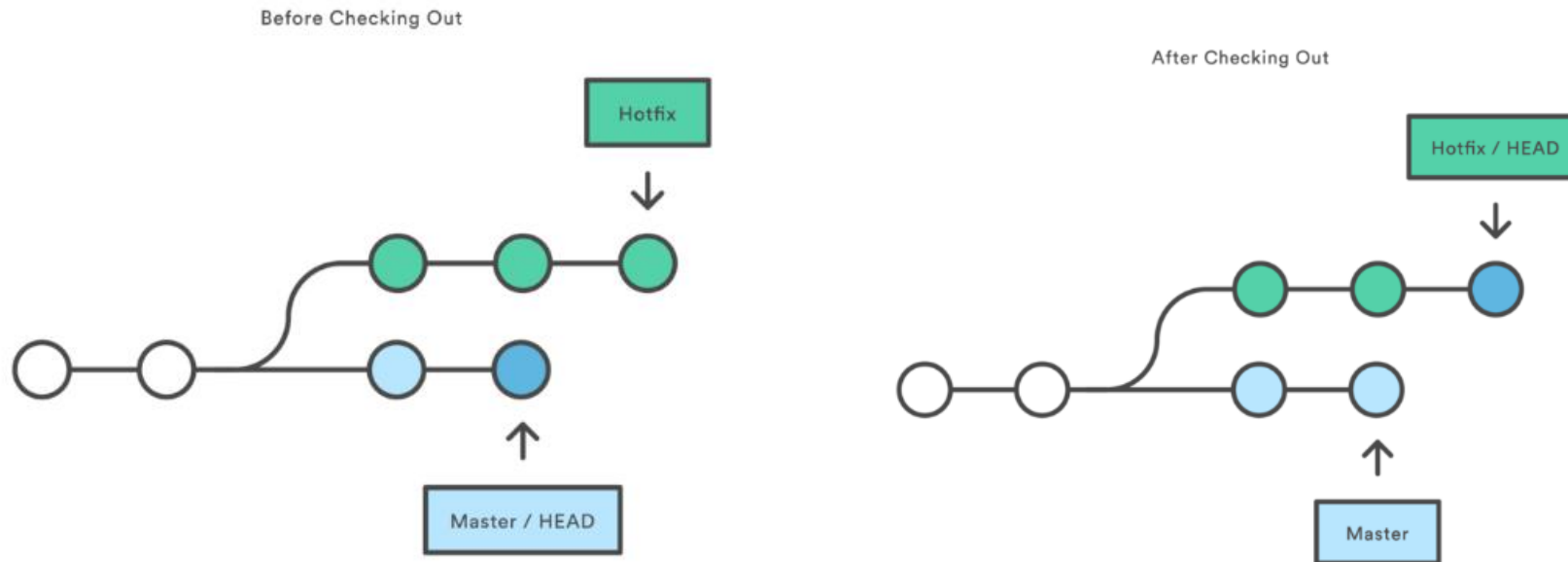
```
* master
```

```
$ git checkout feature-branch
```

```
Switched to branch 'feature-branch'
```

Git. Branches - Checkout

Checkout – перемещение на другую ветку или коммит.



Git. Branches

А вот короткий вариант записи:

```
$ git checkout -b awesome-feature
```

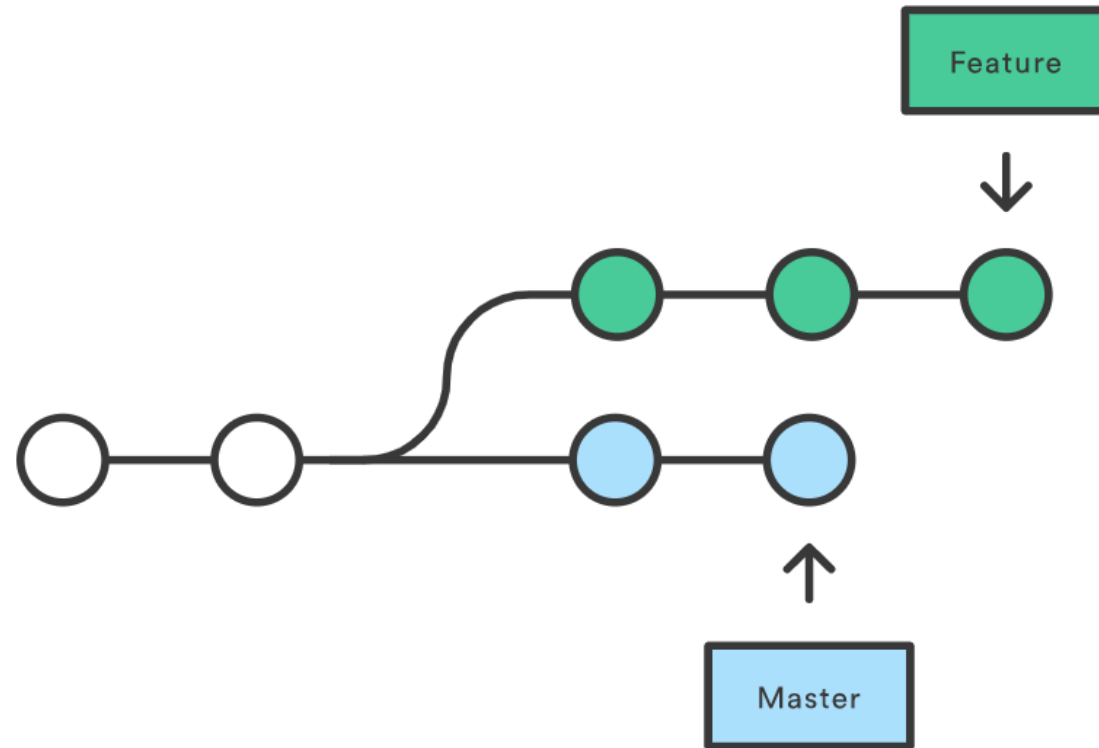
Таким образом Вы создаете новую ветку с именем «awesome-feature» и переходите на нее (создаете на нее указатель).

```
$ git checkout -b awesome-feature
```

```
Switched to a new branch 'awesome-feature'
```

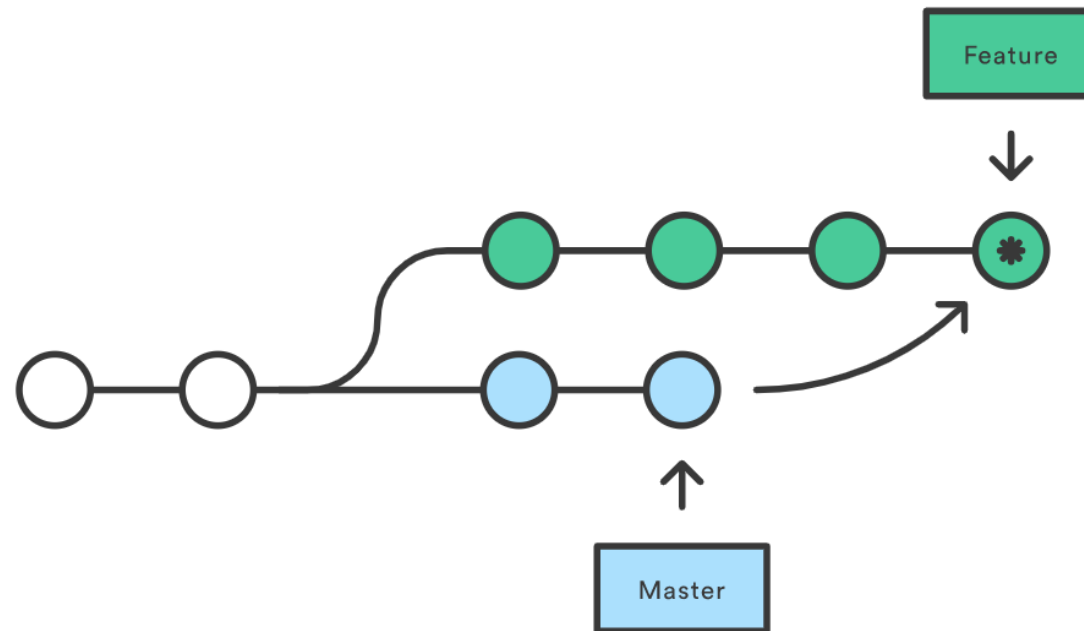
Git. Branches

A forked commit history



Git. Branches – Merge Branches

Когда Вы понимаете, что ваша работа над задачей окончена, наступает момент слить все изменения в основную ветку проекта.



* Merge Commit

Git. Branches – Merge Branches

Для этого Вам понадобится команда:

```
$ git merge <your_branch_name_here>
```

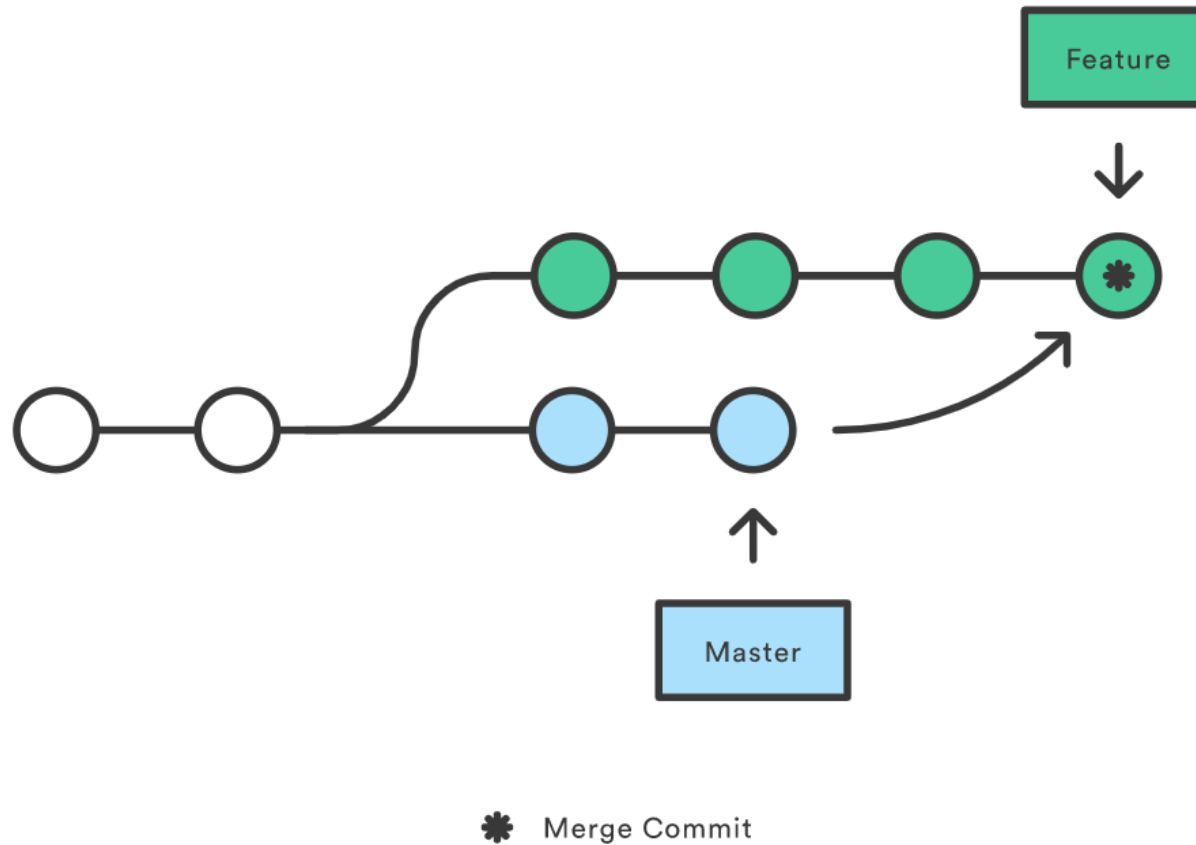
Для этого вы переключаетесь в ветку, в которую хотите собирать изменения, и указываете другую ветку, из которой эти изменения Вы хотите забрать.

```
$ git checkout feature
```

```
$ git merge master
```

```
$ git merge master feature
```

Git. Branches – Merge Branches



Git. Branches - Rebase

Иногда возникает ситуация, когда изменения нужно не слить, а именно перенести из одной ветки в другую.

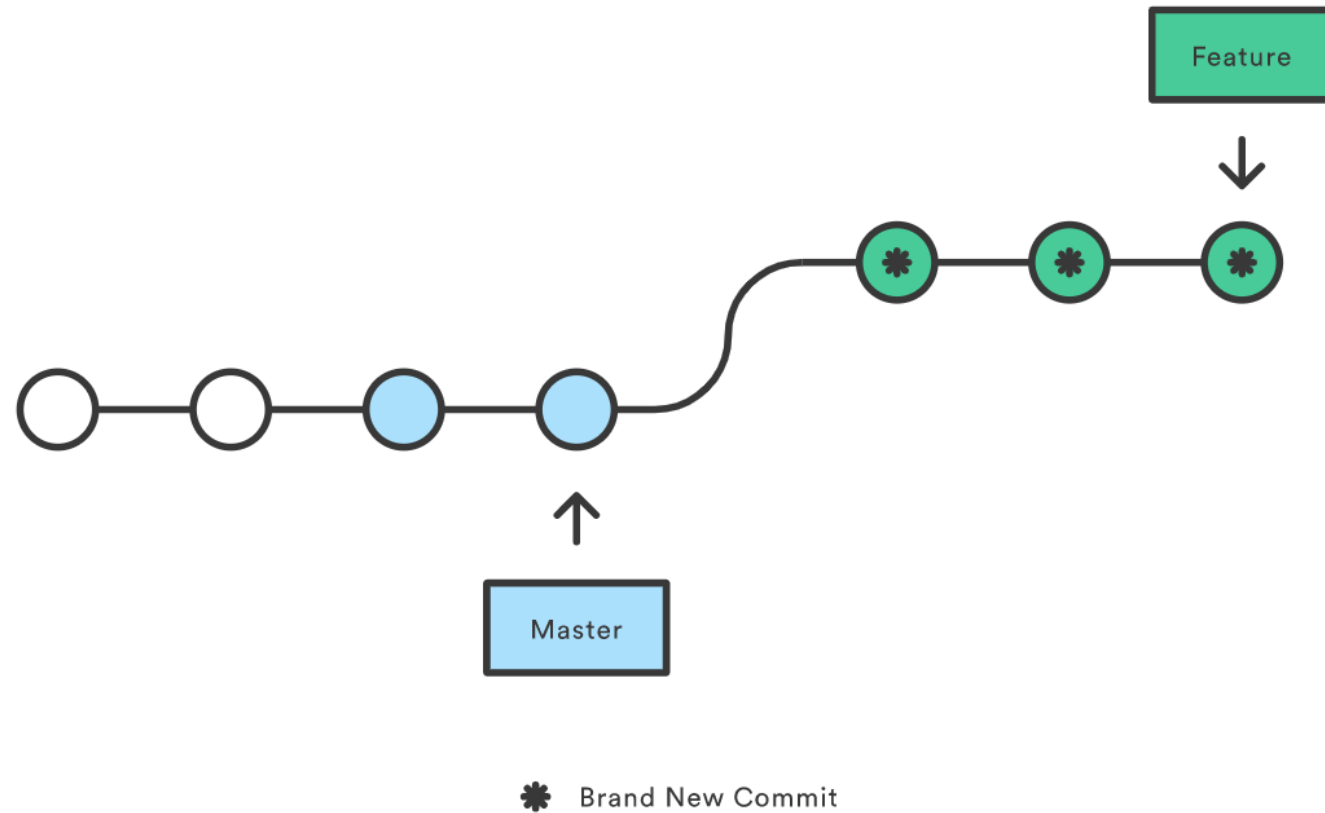
Для этой задачи применяется команда:

```
$ git rebase <your_branch_name_here>
```

```
$ git checkout feature  
$ git rebase master
```

Git. Branches - Rebase

Rebasing the feature branch onto master



Git. Branches - Rebase

Преобразование позволяет сделать дерево коммитов более линейным и аккуратным. Но при этом, команда **rebase** переписывает историю коммитов.

Но его **строго не рекомендуется** часто использовать.

Когда **НЕ** использовать **\$ git rebase**:

- Если ветка является публичной и часто используется в проекте, т.к. переписывание общих веток будет мешать работе других членов команды.
- Когда важна точная история коммитов ветки.



Git

Merging vs. rebasing onto a remote branch

Collaborating on the same feature branch

Merge

Merging

Feature

- ✱ Merge Commit

Rebasing

Feature

- ✿ Brand New Commits

Rebase

Git. Работа с удаленным сервером

Чтобы добавить файлы на удаленный сервер, нужно выполнить **3 операции**:

1. Проиндексировать изменения;
2. Закоммитить изменения;
3. И только затем отправить их на сервер.

Чтобы добавить удаленный репозиторий, Вам поможет команда:

```
$ git remote add origin *ссылка*,
```

где **origin** — имя основного удаленного репозитория (*так его принято называть*).

Если Вы выполняете ***\$ git clone *ссылка****, то **origin** прописывается автоматически.

Git. Pull

Как скачать изменения с сервера (получить изменения из удаленного репозитория)?

Предположим, кто-то ещё работает над теми же файлами, что и Вы. И этот кто-то вносит туда какие-то свои изменения. Чтобы начать работать с уже измененным репозиторием, Вам нужно получить изменения.

Поэтому в данном случае не рекомендуется использовать команду **\$ git clone**, т.к. в таком случае Вам придется заново копировать весь проект. Вам достаточно получить только те фрагменты файлов, которые изменились.

Реализовать это можно с помощью команды:

\$ git pull origin master - забирает и сливает коммиты и ветки.

Но эта команда скачивает только ту ветку, которую Вы указали — ветку **master**.

Если появились новые ветки и коммиты вне основной ветки, то они **не будут загружены на компьютер**.

Git. Push

Проиндексированные файлы, **новые коммиты**, и даже **новые ветки** будут доступны только на Вашем рабочем компьютере, пока Вы не отправите (запустите) все изменения на удаленный сервер (например, на GitHub).

И только после этого их можно будет увидеть на Вашем аккаунте.

Чтобы отправить изменения в удаленный репозиторий:

```
$ git push -u origin master
```

Git. Fetch

Как скачать коммиты и ветки с сервера?

`$ git fetch origin` (только забирает коммиты и ветки)

Это нужно в том случае, если в репозитории на GitHub были внесены новые ветки и коммиты с другого компьютера. Если не забрав эти изменения на свой компьютер, попробовать закоммитить их в новую ветку, то ничего не получится.

Команда `$ git pull` эквивалентна комбинации `$ git fetch` и `$ git merge`.

```
$ git pull origin master
```

```
$ git fetch origin
```

```
$ git merge origin master
```

Git. Push

After merge:

```
$ git push
```

After rebase:

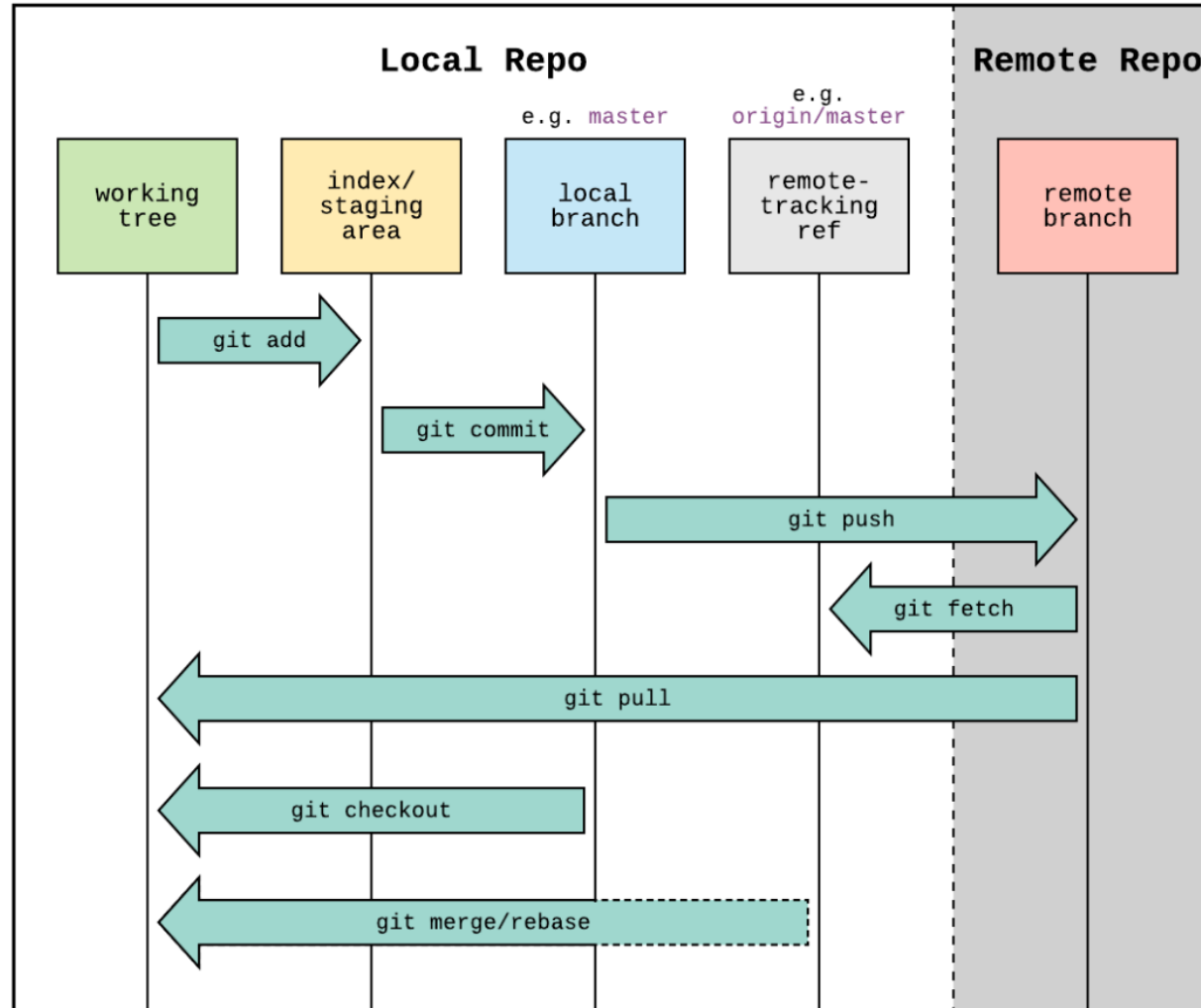
```
$ git push --force
```

* Be very careful with **--force** flag

* BE CAREFUL WITH -- **FORCE** FLAG



Git. Repo workflow



Git. **Revert**

Команда **revert** позволяет Вам отменить последний коммит:

```
$ git revert <commit>
```

При этом команда **revert** оставляет историю, которая показывает, что был **и оригинальный, и отмененный** коммит.



Git. **Reset**

В случае, если Вам необходимо откатиться к предыдущей версии проекта и удалить последний коммит, вы можете воспользоваться командой **reset**:

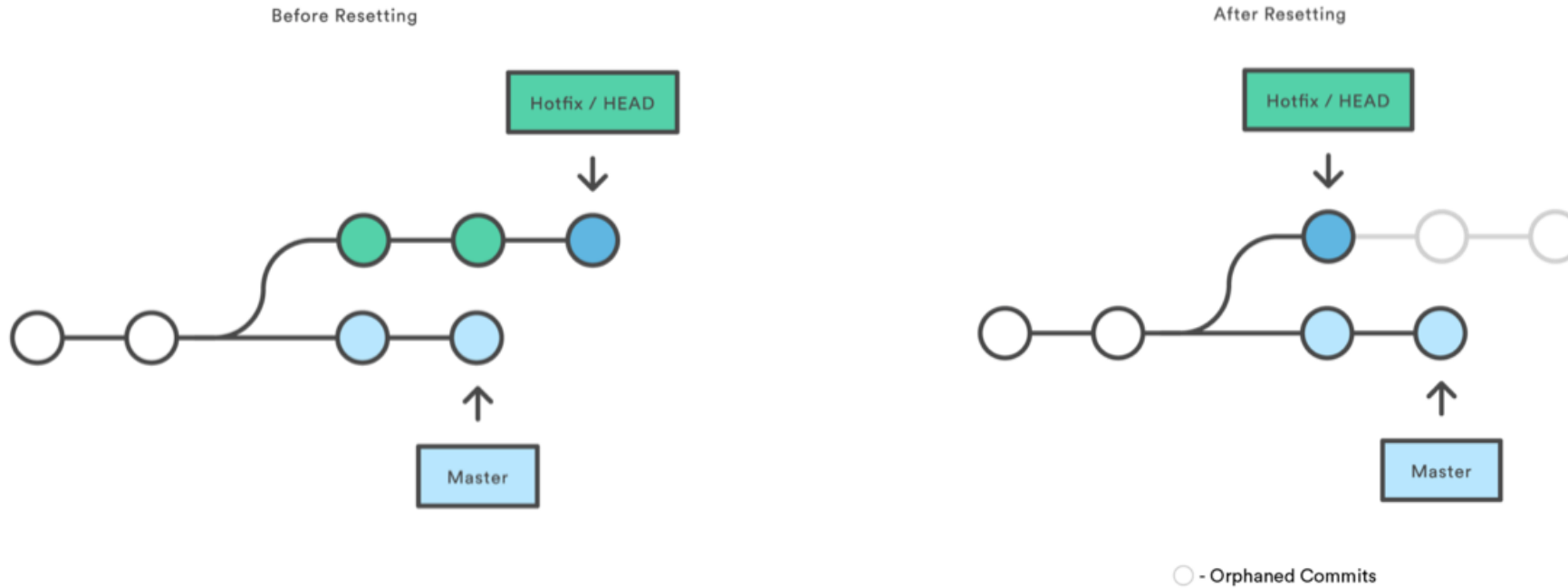
```
$ git reset <commit>
```

Обратите внимание!

После команды **reset** этот коммит не будет отображаться в истории.

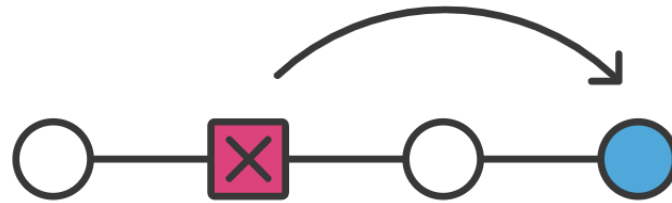
Git. Reset

```
$ git checkout hotfix  
$ git reset HEAD~2
```

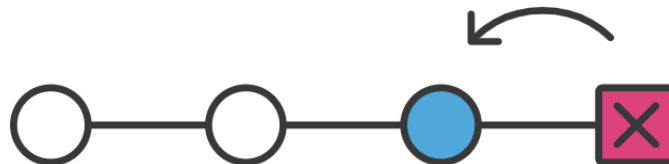


Git. Reset VS Revert

Reverting



Resetting



Git. Conflicts

Конфликты возникают тогда, когда один и тот же файл, в одном и том же месте, но в разных ветках, изменялся по-разному.

Например, в ветке **master** первая строка была изменена, а в ветке **feature** — удалена. При слиянии таких веток возникнет конфликт.

Git сам не знает, какие изменения правильные, а какие - нет.

Поэтому решать, какие изменения верные, предстоит решать Вам.

Но могут возникнуть варианты.

В случае конфликта, при слиянии веток в консоль выведется информация:

```
CONFLICT (content): merge conflict in index.js
```

Git. Conflicts

Как разрешить конфликты?

1. Исправить конфликты и закоммитить.

Для этого Вы вручную редактируете Ваш файлы и сами решаете, что оставить, а что - нет.

```
<html>
  <head>
    <<<<<< HEAD
      <link rel="stylesheet" type="text/css" href="css/style.css" />
    =====
      <link type="text/css" rel="stylesheet" href="style.css" />
    >>>>>> master
  </head>
  <body>
    <h1>Hello, World!</h1>
  </body>
</html>
```

А затем закоммитить изменения: `$ git commit -m "Решил конфликт при слиянии веток"`

2. Отменить слияние.

`$ git merge --abort`

Git. Pull before Push!

При работе в команде Вам необходимо выполнить **3 этапа**:

1. Сначала забрать все изменения с сервера при помощи **\$ git pull;**
2. Разрешить все конфликты, которые могли возникнуть;
3. И только потом пушить изменения на сервер через **\$ git push.**

И должно быть только так, и никак иначе!

Полезные источники по Git

- Подробный учебник по Git на русском языке:

<https://git-scm.com/book/ru/v2>

- Умение работы с командной строкой определяет, насколько Вы уверенно и свободно сможете пользоваться Git. Вот курс по командной строке (Bash):

<https://www.codecademy.com/learn/learn-the-command-line>

- Визуальное приложение, которое наглядно поможет понять основы работы с Git, в частности с ветками:

<https://learngitbranching.js.org/>

- Подробные видео-уроки по Git (на примере JavaScript):

<http://learn.javascript.ru/screencast/git>

Полезные источники по Git

- Спецификация по оформлению коммитов:
<https://www.conventionalcommits.org/en/v1.0.0-beta.2/#summary>
- Развернутый ответ на то, чем отличается Merge от Rebase:
<https://www.atlassian.com/git/tutorials/merging-vs-rebasing>
- Чем отличается Reset от Revert:
<https://www.atlassian.com/git/tutorials/resetting-checking-out-and-reverting>

Что нужно для сдачи зачета?

1. **Создать свой GitHub/Gitlab/Bitbucket профиль** под своим именем и фамилией (также следует указать **факультет/курс/специальность/группу**);
2. **Создать удаленный репозиторий** для каждой лабораторной работы;
3. **Сделать коммиты** к каждой лабораторной работе и **запустить** их в Ваш репозиторий;
4. Быть готовым **показать Ваш профиль** с выполненными и запущенными лабораторными вашим преподавателям, включая меня.
5. Быть готовым рассказать, **как работает Git**, и продемонстрировать **умение работать с Git**.

**Я настоятельно не рекомендую
пользоваться сторонними
клиентами для работы с Git!**

Вся ваша работа с Git в рамках лабораторных работ должна вестись только через консоль!

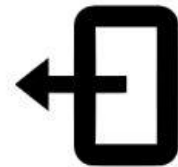
In case of fire



1. `git commit`



2. `git push`



3. leave building

Спасибо за внимание!