

# Системы контроля версий



Андрей  
Борю



## Андрей Борю

Principal DevOps Engineer, Snapcart





# План занятия

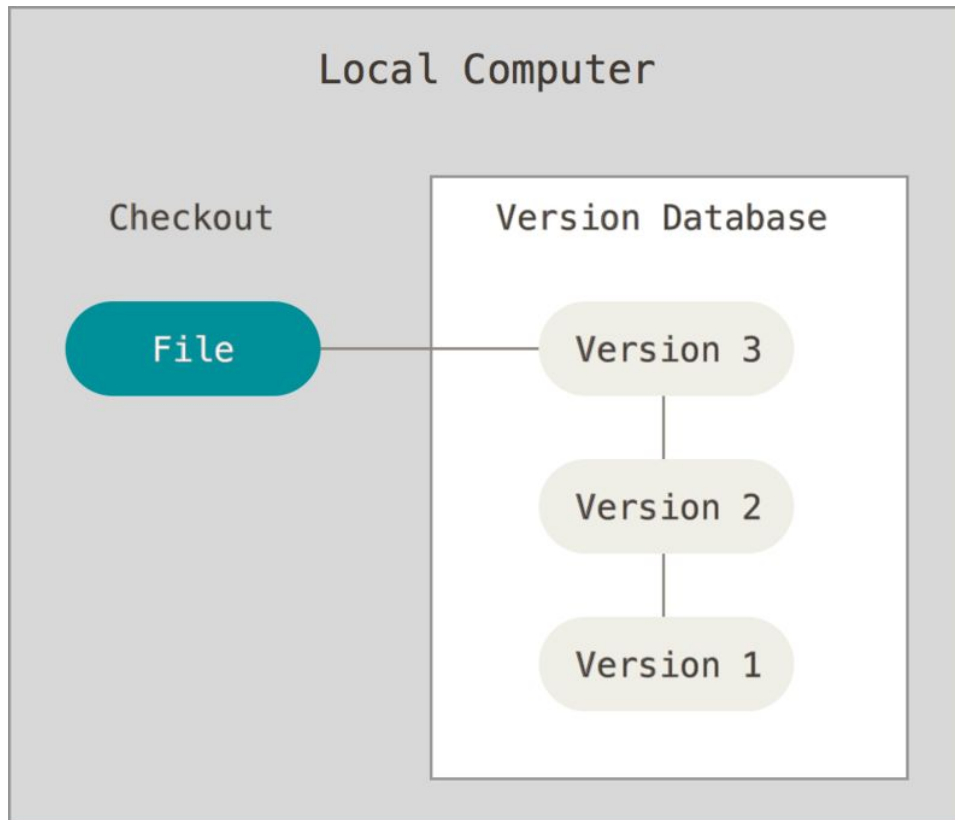
1. [Системы контроля версий](#)
2. [Краткая история Git](#)
3. [Основы Git](#)
4. [Первоначальная настройка](#)
5. [Создание репозитория](#)
6. [Запись изменения в репозиторий](#)
7. [Итоги](#)
8. [Домашнее задание](#)



# Системы контроля версий

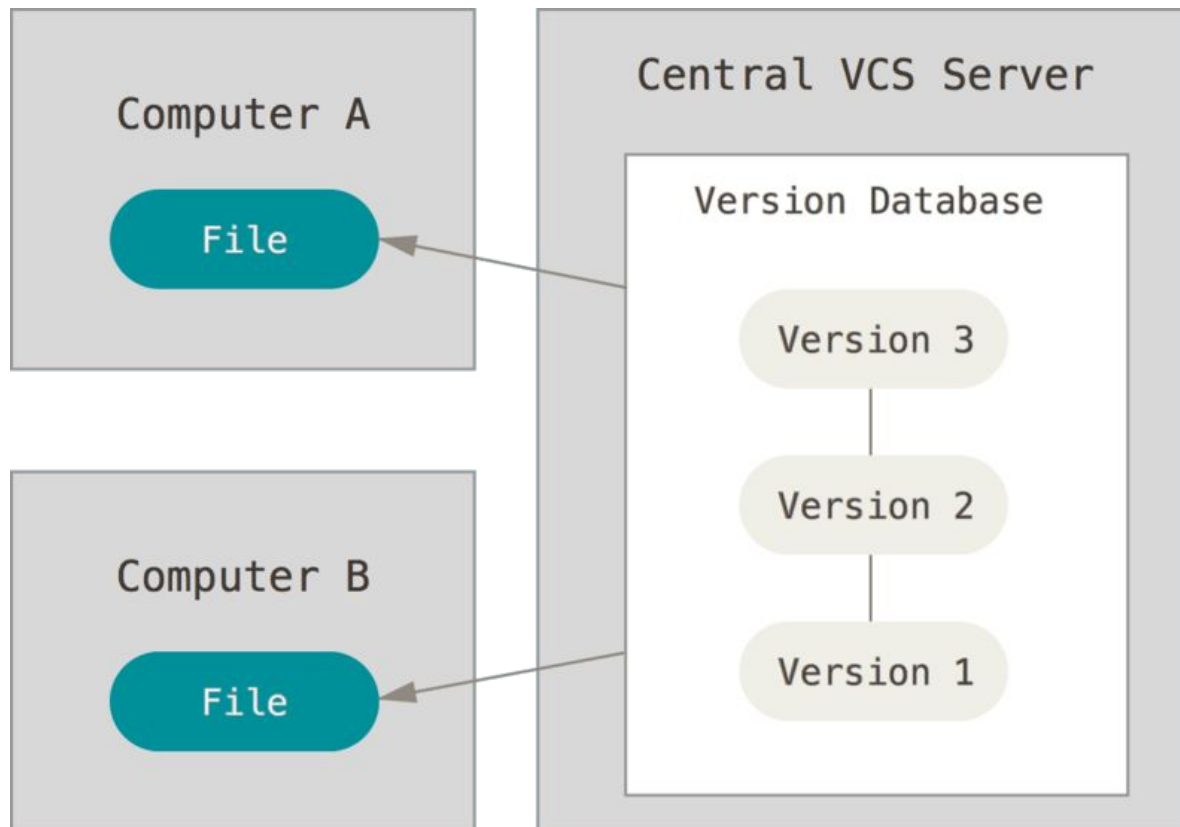
# Локальные VCS

Например, Revision Control System (RCS)



# Централизованные VCS

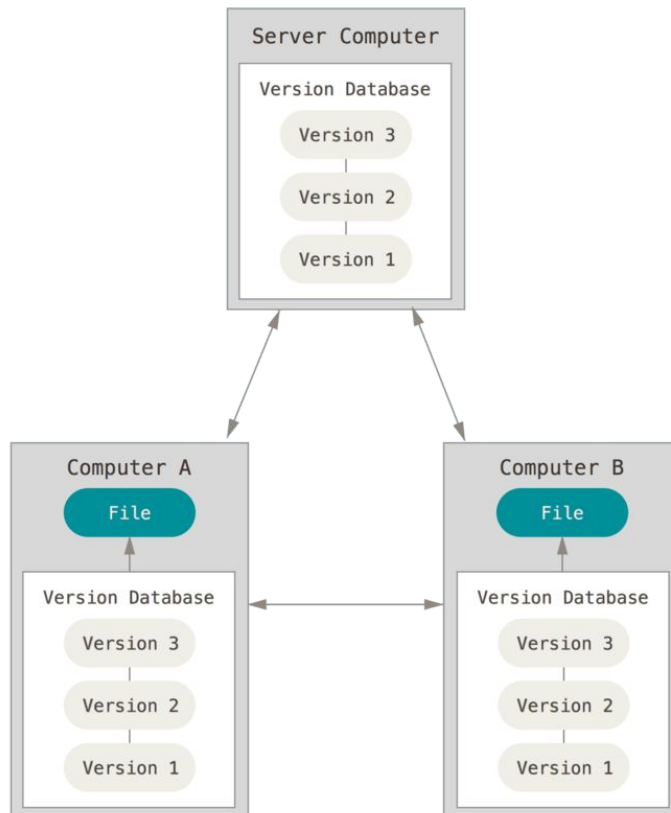
Например, Subversion и Perforce



Взято с сайта: [git-scm.com](http://git-scm.com)

# Распределённые VCS

Например, Git, Mercurial, Bazaar и Darcs





# Краткая история Git





# Краткая история Git

- **Ядро Linux** - большой проект с открытым кодом
- **1991-2002** - изменения передавались в виде патчей и архивов
- **2002-2005** - использование децентрализованной VCS BitKeeper
- **2005** - разработка собственной утилиты, основанные на работе с VCS BitKeeper



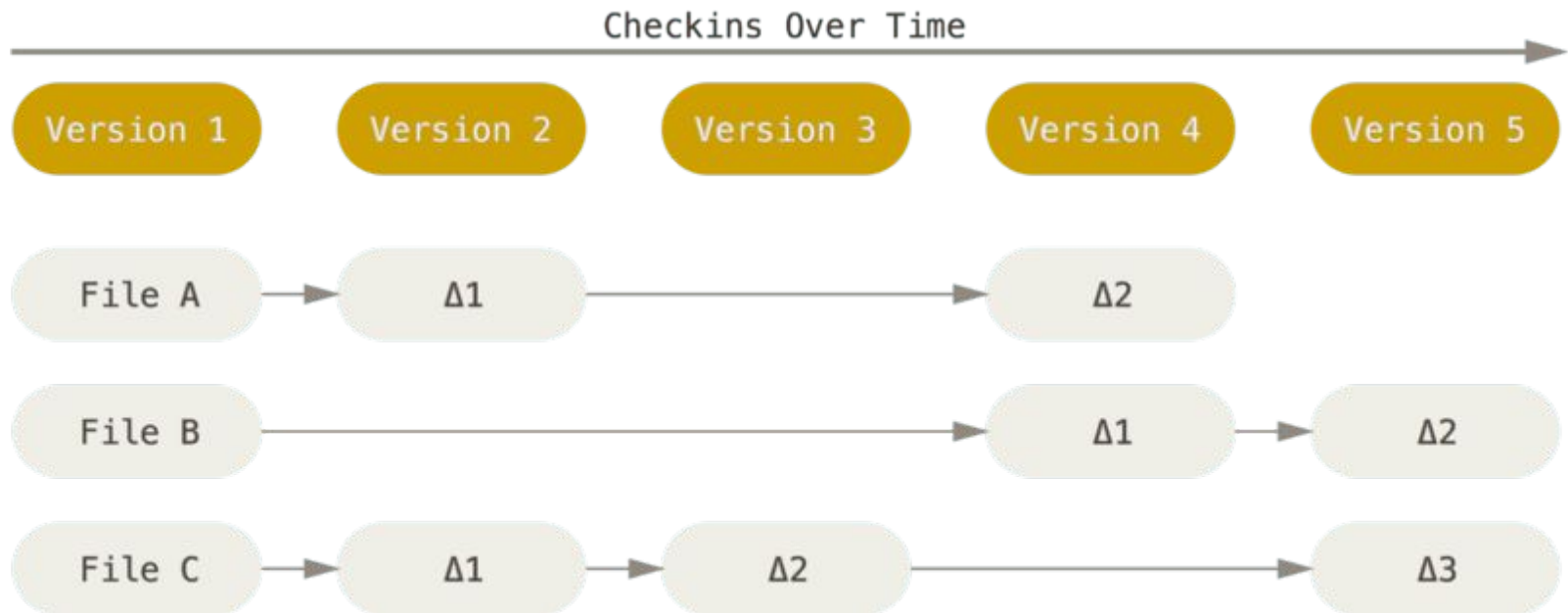
## Какие проблемы должен был решить Git?

- Скорость
- Простая архитектура
- Хорошая поддержка нелинейной разработки (тысячи параллельных веток)
- Полная децентрализация
- Возможность эффективного управления большими проектами

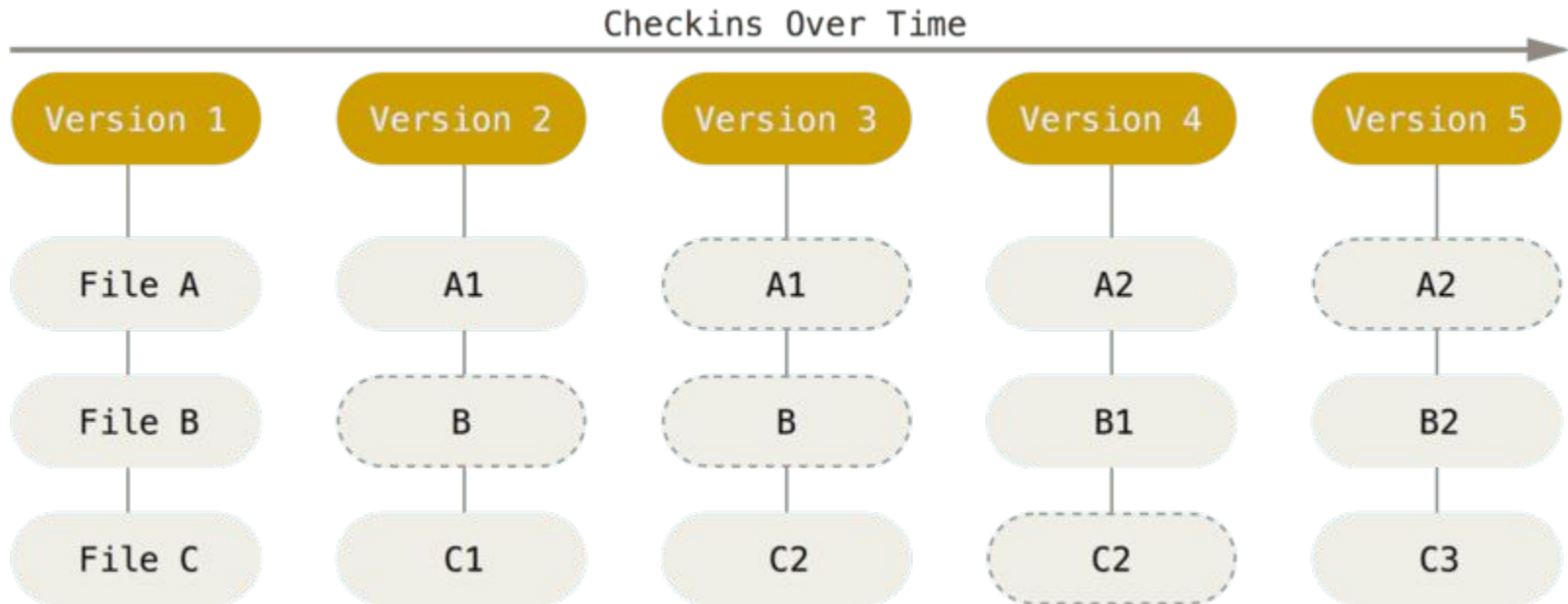


# Основы Git

# Хранение различий



# Хранение снимков



---

## Особенности Git

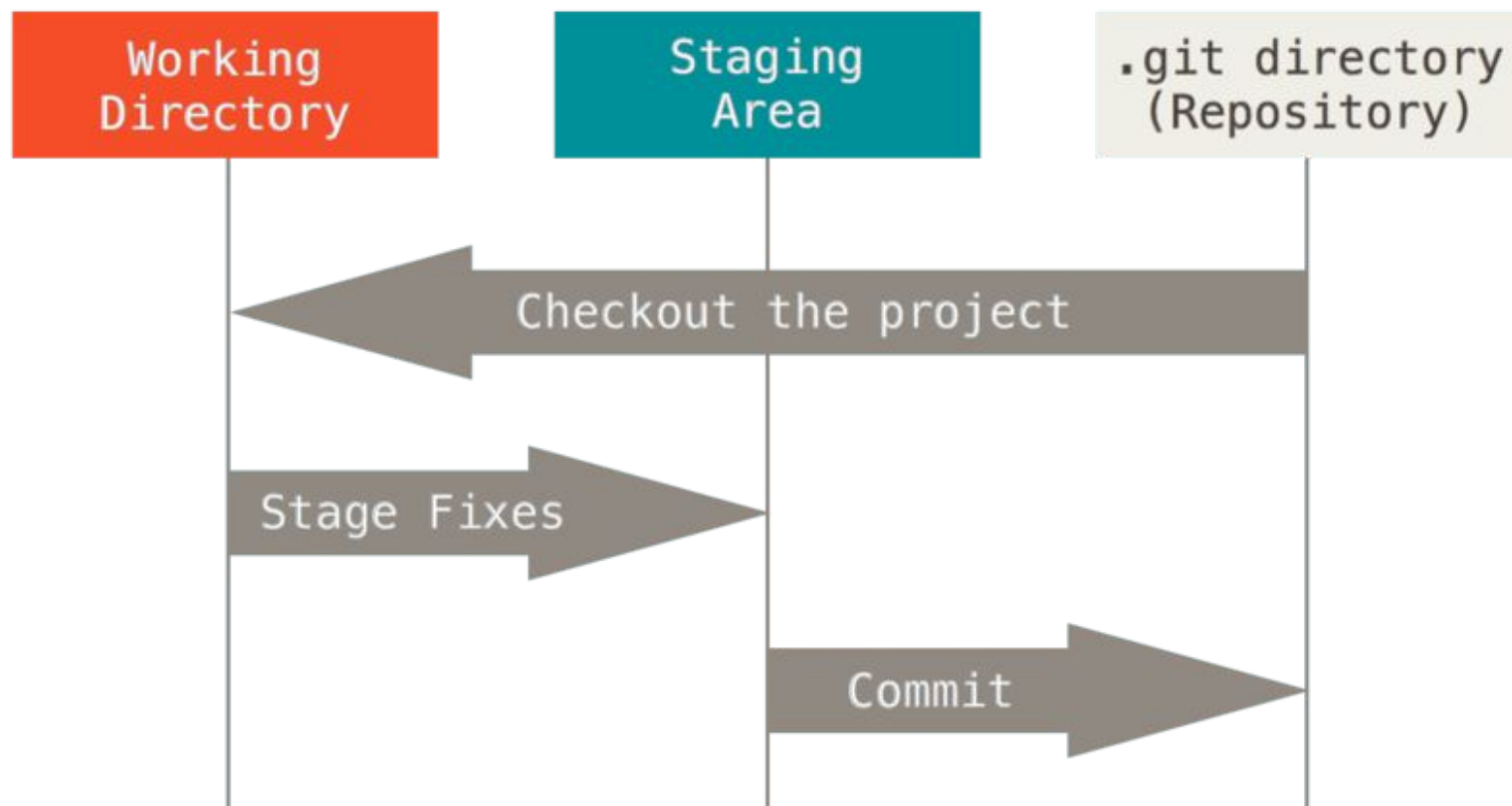
- **Операции выполняются локально** - системе не нужна никакая информация с других компьютеров в вашей сети
- **Целостность** - в Git для всего вычисляется хеш-сумма, и только потом происходит сохранение. Невозможно изменить содержимое файла или директории так, чтобы Git не узнал об этом
- **Обычно только добавляет данные** - при выполнении каких-либо действий в Git, практически все из них только добавляют новые данные в базу. Мы можем экспериментировать, не боясь серьёзных проблем



## Три состояния файла в Git

- **Зафиксированный** (committed) - файл уже сохранён в вашей локальной базе
- **Измененный** (modified) - файлы, которые поменялись, но ещё не были зафиксированы
- **Подготовленный** (staged) - изменённые файлы, отмеченные для включения в следующий коммит.

## Три состояния файла в Git







## Базовый подход к работе с Git

- Изменяете файлы в рабочей директории
- Выборочно добавляете файлы в индекс изменения для следующего коммита
- Сохраняете файлы из индекса изменений в Git директорию

---

# Командная строка

**Командная строка** - единственное место, где можно запустить *все* команды Git. Большинство клиентов с графическим интерфейсом реализуют для простоты только некоторую часть функциональности

# Установка

## Linux:

- `sudo dnf install git-all` (Fedora, CentOS)
- `sudo yum install git-all` (Fedora, CentOS)
- `sudo apt install git` (Debian, Ubuntu)
- <https://git-scm.com/download/linux>

## MacOs:

- `brew install git` (используя менеджер пакетов brew)
- `git --version` (запустится установщик xCode)
- <https://git-scm.com/download/mac>

## Windows:

- <http://git-scm.com/download/win>



# Первоначальная настройка Git

# Команда `git config`

**Git config** - утилита, которая позволяет просматривать и настраивать параметры, контролирующие все аспекты работы Git, а также его внешний вид.

**Параметры сохранены в трех местах:**

- `/etc/gitconfig`, опция `--system`
- `~/.gitconfig` или `~/.config/git/config`, опция `--global`
- `.git/config`, опция `--local`

**Посмотреть все настройки и где они заданы:**

```
git config --list --show-origin
```

# Первоначальная настройка

Первоначальная настройка - указать имя и email.

## Установить имя и email:

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

Далее выбрать редактор, который будет использоваться в Git.

## Установить отличный от стандартного редактор:

- `git config --global core.editor emacs*`

\* например, Emacs

# Как получить помощь?

Если вам нужна помощь при использовании Git, есть три основных способа сделать это:

- `git help команда`
- `git команда --help`
- `man git-команда`

Например:

- `git help add`
- `git add --help`
- `git add -h`
- `man git-add`



# Промежуточный итог





## Что мы узнали

- Что такое Git
- Чем Git отличается от централизованных VCS
- Как настроить и персонализированную рабочую версию Git в вашей ОС




# Создание репозитория

# Создание репозитория в существующей директории

- `cd ~/your_project` - перейти в каталог будущего проекта
- `git init` - создать в текущей директории новую поддиректорию с именем `.git`
- `echo "my_rules" > CONTRIBUTING.md`
- `git add *` - добавить под версионный контроль существующие файлы
  - или `git add CONTRIBUTING.md`
- `git commit -m "first commit"` - выполнить коммит

# Клонирование существующего репозитория

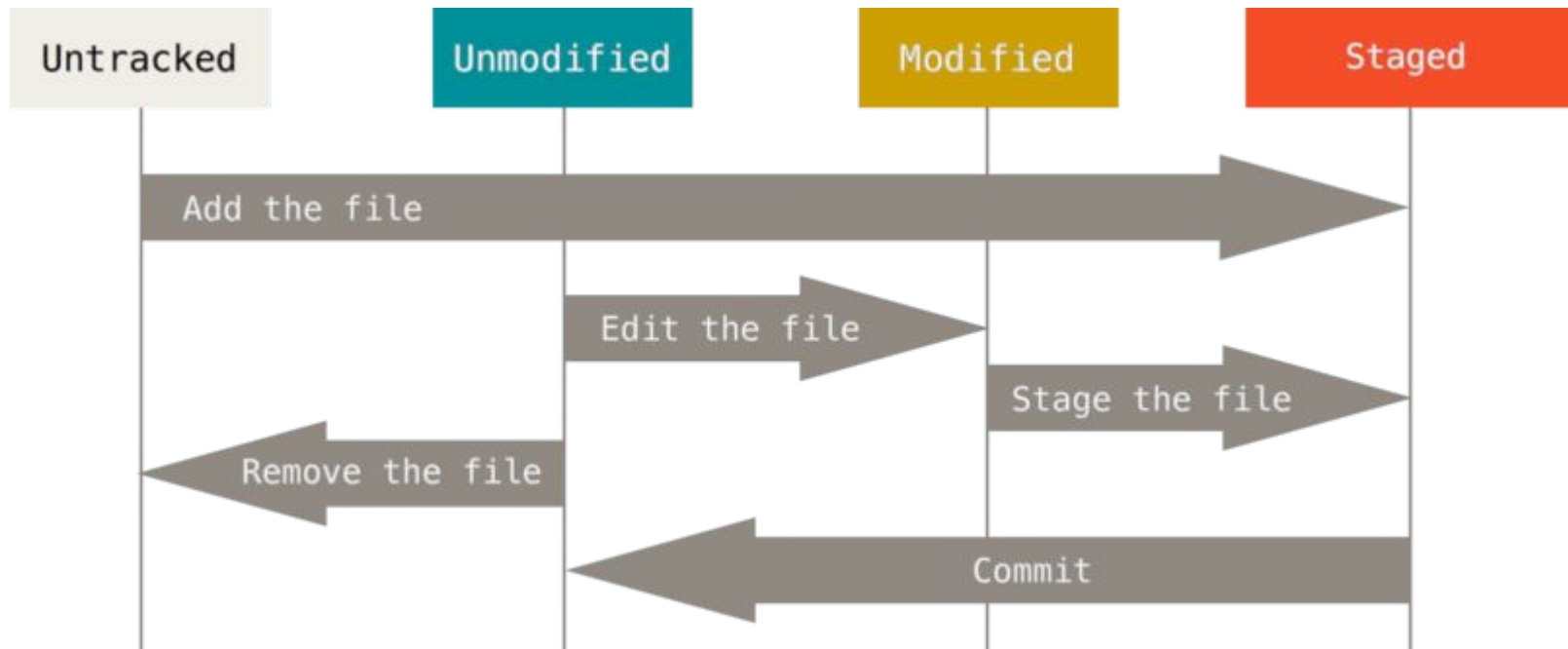
- `git clone https://github.com/libgit2/libgit2 my_dir`  
или
- `git clone git@github.com:libgit2/libgit2.git my_dir` - получить копию существующего Git-репозитория
- `cd my_dir` - перейти в созданный при клонировании каталог
- `git add *` - добавить файлы под кодтроль гита
- `git commit -m "first commit"` - выполнить КОММИТ



# Запись изменений в репозиторий

# Жизненный цикл файла

**Untracked** - неотслеживаемый. **Unmodified** – не измененный. **Modified** – измененный. **Staged** - зафиксированный.



## Определение состояния файла

Команда **git status** - основной инструмент, используемый для определения, какие файлы в каком состоянии находятся. После клонирования вы увидите что-то вроде этого:

```
$ git status
```

```
On branch master
```

```
Your branch is up-to-date with 'origin/master'.
```

```
nothing to commit, working directory clean
```

## Добавляем файл

Предположим, вы добавили в свой проект новый файл простой файл README. Если этого файла раньше не было, и вы выполните **git status** - вы увидите свой неотслеживаемый файл вот так:

```
$ echo "My Project" > README
```

```
$ git status
```

```
On branch master
```

```
Your branch is up-to-date with 'origin/master'.
```

```
Untracked files:
```

```
(use "git add <file>..." to include in what will be committed)
```

```
README
```

```
nothing added to commit but untracked files present (use "git add" to track)
```



## Отслеживание новых файлов

Для того, чтобы начать отслеживать (добавить под версионный контроль) новый файл, используется команда **git add**. Чтобы начать отслеживание файла README, вы можете выполнить следующее:

```
$ git add README
```

```
$ git status
```

```
On branch master
```

```
Your branch is up-to-date with 'origin/master'.
```

```
Changes to be committed:
```

```
(use "git restore --staged <file>..." to unstage)
```

```
new file:   README
```

# Индексация измененных файлов

Давайте модифицируем файл, уже находящийся под версионным контролем. Если вы измените отслеживаемый файл CONTRIBUTING.md и после этого снова выполните команду **git status**, то результат будет примерно следующим:

```
$ echo "new line" >> CONTRIBUTING.md
```

```
$ git status
```

```
On branch master
```

```
Your branch is up-to-date with 'origin/master'.
```

```
Changes to be committed:
```

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

```
new file: README
```

```
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: CONTRIBUTING.md
```

# Индексация измененных файлов

Выполним **git add**, чтобы проиндексировать CONTRIBUTING.md, а затем снова выполним **git status**.

```
$ git add CONTRIBUTING.md
```

```
$ git status
```

```
On branch master
```

```
Your branch is up-to-date with 'origin/master'.
```

```
Changes to be committed:
```

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

```
new file: README
```

```
modified: CONTRIBUTING.md
```

# Изменение после индексации

Вы открываете файл, вносите и сохраняете необходимые изменения и вроде бы готовы к коммиту. Но давайте-ка ещё раз выполним **git status**.

```
$ echo "second line" >> CONTRIBUTING.md
```

```
$ git status
```

On branch master

Your branch is up-to-date with 'origin/master'.

Changes to be committed:

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

new file: README

modified: CONTRIBUTING.md

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: CONTRIBUTING.md

## Сокращенный вывод статуса

Вывод команды **git status** довольно всеобъемлющий и многословный. Если вы выполните **git status -s** или **git status --short**, вы получите гораздо более упрощенный вывод:

**\$ git status -s ИЛИ git status --short**

M README – модифицирован, не индексирован  
MM Rakefile – модифицирован, индексирован и еще раз модифицирован  
A lib/git.rb – добавленный в отслеживаемые  
M lib/simplegit.rb – модифицирован, индексирован  
?? LICENSE.txt – новый неотслеживаемый

# Игнорирование файлов

```
$ cat .gitignore
```

```
# Исключить все файлы с расширение .a
```

```
*.a
```

```
# Но отслеживать файл lib.a даже если он подпадает под исключение выше  
!lib.a
```

```
# Исключить файл TODO в корневой директории, но не файл в subdir/TODO  
/TODO
```

```
# Игнорировать все файлы в директории build/  
build/
```

```
# Игнорировать файл doc/notes.txt, но не файл doc/server/arch.txt  
doc/*.txt
```

```
# Игнорировать все .txt файлы в директории doc/  
doc/**/*.*.txt
```

# Правила шаблонов

- Пустые строки, а также строки, начинающиеся с #, игнорируются
- Стандартные шаблоны являются глобальными и применяются рекурсивно для всего дерева каталогов
- Чтобы избежать рекурсии используйте символ слэш (/) в начале шаблона
- Чтобы исключить каталог добавьте слэш (/) в конец шаблона
- Можно инвертировать шаблон, используя восклицательный знак (!) в качестве первого символа

# Правила шаблонов

- Символ (\*) соответствует 0 или более символам
- Последовательность [abc] — любому символу из указанных в скобках (в данном примере a, b или c)
- Знак вопроса (?) соответствует одному символу
- Квадратные скобки, в которые заключены символы, разделённые дефисом ([0-9]), соответствуют любому символу из интервала
- Две звёздочки указывают на вложенные директории: a/\*\*/z соответствует a/z, a/b/z, a/b/c/z, и так далее



---

# Правила шаблонов

- Коллекция примеров: <https://github.com/github/gitignore>
- Подробнее: `man gitignore`
- Файлы `.gitignore` могут находится в разных каталогах

## Просмотр изменений

- `git diff` - показывает что изменили, но пока не проиндексировали
- `git diff --staged` - сравнивает проиндексированные изменения с последним коммитом
- `git diff --staged` и `git diff --cached` - синонимы
- `git difftool` - чтобы использовать графическую утилиту

# Просмотр изменений

**\$ git diff --cached**

```
diff --git a/CONTRIBUTING.md b/CONTRIBUTING.md
```

```
index 8ebb991..643e24f 100644
```

```
--- a/CONTRIBUTING.md
```

```
+++ b/CONTRIBUTING.md
```

@@ -65,7 +65,8 @@ branch directly, things can get messy.

Please include a nice description of your changes when you submit your PR; if we have to read the whole diff to figure out why you're contributing in the first place, you're less likely to get feedback and have your change merged in.

+merged in. Also, split your changes into comprehensive chunks if you patch is +longer than a dozen lines.

If you are starting to work on a particular area, feel free to submit a PR that highlights your work in progress (and note in the PR title that it's

# Коммит изменений

Теперь, когда ваш индекс находится в таком состоянии, как вам и хотелось, вы можете зафиксировать свои изменения.

```
$ git commit
```

```
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
# Your branch is up-to-date with 'origin/master'.
#
# Changes to be committed:
#   new file:   README
#   modified:   CONTRIBUTING.md
#
~
".git/COMMIT_EDITMSG" 9L, 283C
```

---

## Коммит изменений

- `git commit -m "add readme"` - быстрый способ создать коммит
- `git commit -v` - добавить в комментарий коммита diff
- `git config --global core.editor` - изменить редактор текста коммита

# Коммит изменений

Есть и более быстрый способ выполнить коммит без открытия внешнего текстового редактора:

```
$ git commit -m "[STORY-182] Fix index page"
```

```
[master 463dc4f] [STORY-182] Fix index page  
2 files changed, 2 insertions(+)  
create mode 100644 README
```

# Игнорирование индексации

Если у вас есть желание пропустить этап индексирования, Git предоставляет простой способ.

```
$ git status
```

```
On branch master
```

```
Your branch is up-to-date with 'origin/master'.
```

```
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:   CONTRIBUTING.md
```

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

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

```
[master 83e38c7] added new benchmarks
```

```
1 file changed, 5 insertions(+), 0 deletions(-)
```

# Удаление файлов

Чтобы удалить файл из Git, не достаточно просто удалить файл из рабочего каталога:

```
$ rm PROJECTS.md
```

```
$ git status
```

```
On branch master
```

```
Your branch is up-to-date with 'origin/master'.
```

```
Changes not staged for commit:
```

```
  (use "git add/rm <file>..." to update what will be committed)
```

```
  (use "git checkout -- <file>..." to discard changes in working directory)
```

```
    deleted:   PROJECTS.md
```

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



## Удаление файлов

Затем, если вы выполните команду **git rm**, удаление файла попадёт в индекс. После следующего коммита файл исчезнет и больше не будет отслеживаться.

```
$ git rm PROJECTS.md
```

```
rm 'PROJECTS.md'
```

```
$ git status
```

```
On branch master
```

```
Your branch is up-to-date with 'origin/master'.
```

```
Changes to be committed:
```

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

```
deleted:    PROJECTS.md
```

## Удаление файлов

- `git rm -f readme` - удалить файл, если он уже проиндексирован
- `git rm --cached readme` - оставить файл на жестком диске, но перестать отслеживать
- `git rm log/\*.log` - возможность использования шаблонов

# Перемещение файлов

Для перемещения файлов необходимо также сообщить об этом гиту, а не просто переместить файлы в рабочем каталоге:

```
$ git mv README.md README
```

```
$ git status
```

```
On branch master
```

```
Your branch is up-to-date with 'origin/master'.
```

```
Changes to be committed:
```

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

```
renamed: README.md -> README
```

# Перемещение файлов

Git mv можно заменить такой, более длинной, последовательностью команд:

```
$ mv README.md README  
$ git rm README.md  
$ git add README
```



# Итоги



## Чему мы научились

- Создавать и клонировать репозитории
- Определять и изменять состояния файлов
- Записывать изменения в репозиторий
- Удалять и перемещать файлы в репозитории



# Домашнее задание



# Домашнее задание

Давайте посмотрим ваше [домашнее задание](#).

- Вопросы по домашней работе задавайте **в чате** мессенджера Slack.
- Задачи можно сдавать **по частям**.
- Зачёт по домашней работе проставляется после того, как **приняты все задачи**.



**Задавайте вопросы и  
пишите отзыв о лекции!**

**Андрей Борю**



[andreyborue](https://t.me/andreyborue)



[andreyborue](https://netology.ru/andreyborue)



[andreyborue](https://t.me/andreyborue)