

# Дорожная карта

- 0. Приступаем
- 1. Введение в Git
- 2. Начало работы с Git ➡
- 3. Просмотр истории
- 4. Ветвление
- 5. Слияние
- 6. Отмена изменений
- 7. Рабочий процесс

## Дорожная карта (продолжение)

- 8. Работа в команде
- 9. Метки
- 10. Последние штрихи
- 11. Завершаем

# Установка Git

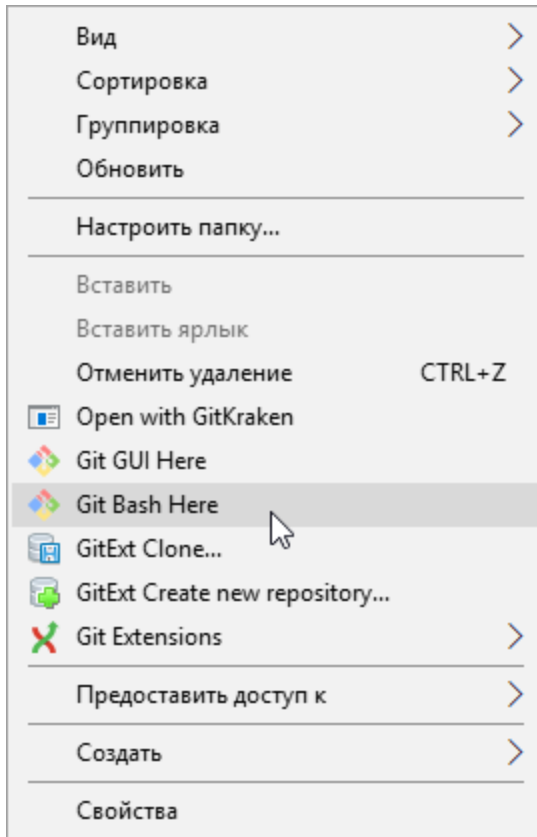
- [Git доступен](#) на многих операционных системах, включая Windows, Linux, MacOS X
- Обычно устанавливается из пакетов и/или инсталляторов

# Командная строка

- Позволяет запустить *все* виды команд.
- После командной строки проще освоить графические клиенты.
- `Bash` - удобная командная оболочка

# Командная строка (продолжение)

Чтобы открыть консоль в нужной папке, в этой папке выбрать `Git Bash Here` в КОНТЕКСТНОМ МЕНЮ:



# Пути к папкам в Windows

В командной строке пути к файлу или папке выглядят несколько иначе:

- `c:\work\myproject` → `/c/work/myproject`
- `/c/Users/Username` → `~`

# Графический Git

GUI в комплекте:

- `gitk`
- `git gui`

Сторонние проекты:

- [Git Extentions](#)
- [GitKraken](#)
- [и много других](#)

😊 Пробуйте разные и находите своё!

# Первоначальная настройка Git

`git config` позволяет настраивать и просматривать параметры.

Установить значение:

```
$ git config [scope] section.var value
```

Прочитать значение:

```
$ git config [scope] section.var
```

Получить все значения:

```
$ git config [scope] --list
```



# Первоначальная настройка Git (продолжение)

Область определения ( `scope` ) настроек:

Опция	На уровне	Пользователь	Репозиторий	Файл
<code>--system</code>	системы	все	все	<code>/etc/gitconfig</code>
<code>--global</code>	пользователя	один	все	<code>~/.gitconfig</code>
<code>--local</code>	репозитория	один	один	<code>.git/config</code>
<code>--file</code> <code>&lt;filename&gt;</code>	команды	один	один	<code>&lt;filename&gt;</code>

Настройки на каждом следующем (более локальном) уровне подменяют настройки из предыдущих (более глобальных) уровней.

## Первоначальная настройка Git (продолжение)

Изменить данные пользователя:

```
$ git config --global user.name "Maxim Suslov"  
$ git config --global user.email msuslov@luxoft.com
```

Проверка:

```
$ cat ~/.gitconfig  
[user]  
    name = Maxim Suslov  
    email = msuslov@luxoft.com
```

## Первоначальная настройка Git (продолжение)

- Выбрать редактор:

```
$ git config --global core.editor "code --wait"
$ git config --global core.editor \
  "'C:/Program Files/Notepad++/notepad++.exe' -multiInst -notabbar -nosession -noPlugin"
$ git config --global core.editor \
  "'C:/Program Files/sublime text 3/subl.exe' -w -n"
```

[Большой список](#) настройки разных редакторов.

# Проверка настроек

Команда `git config --list` покажет все настройки:

```
$ git config --list
user.name=Maxim Suslov
user.email=msuslov@luxoft.com
color.status=auto
color.interactive=auto
...
```

Если ключи появляется несколько раз, то используется последнее значение.

Проверить значение конкретного ключа можно выполнив `git config <section.var> :`

```
$ git config user.name
Maxim Suslov
```

# Практика

Настроить параметры git, используя команды с предыдущих слайдов:

- `user.name`
- `user.email`
- `core.editor`

## Практика (продолжение)

Проверить настройки:

- открыть конфигурационный файл `~/.gitconfig` и увидеть указанные выше параметры
- выполнить команды и проверить вывод

```
$ git config user.name  
$ git config user.email  
$ git config core.editor  
$ git config --list
```

- выполнить команду и посмотреть, что запустился заданный редактор

```
$ git config --global --edit
```

 Когда нужно использовать разные уровни настроек?

# Как получить помощь

Получить помощь по любой команде:

```
$ git help <команда>  
$ git <команда> --help  
$ man git-<команда>
```

Например, открыть руководство по команде `config`:

```
$ git help config
```



# Практика

Получить справку по команде:

```
$ git help config  
$ git config --help
```

 За что отвечает параметр `core.pager` ?

# Создание Git-репозитория


Создание репозитория в существующей директории

```
$ cd <folder>  
$ git init
```

Клонирование существующего репозитория

```
$ git clone <url>
```

# Практика

 Создание репозитория в существующей папке

1. Создайте папку для нового проекта.

```
$ mkdir -p /c/work/myproject  
$ cd /c/work/myproject
```

2. Создайте и/или скопируйте несколько файлов разных типов в эту папку.

3. Инициализируйте новый репозиторий:

```
$ git init
```

 Что изменилось в этой папке?

 Находятся ли файлы в этой папке под версионным контролем?

# Клонирование существующего репозитория

Клонирование репозитория осуществляется командой `git clone <url> [folder]`:

```
$ git clone https://github.com/libgit2/libgit2
$ git clone https://github.com/libgit2/libgit2 mylibgit
```

Различные протоколы передачи `https://` , `git://` , `user@server:path/to/repo.git`

# Частичное клонирование репозитория

Нельзя клонировать одну папку существующего репозитория.

Можно скопировать заданное число последних коммитов ( `--depth <n>` )

```
$ git clone https://github.com/libgit2/libgit2 --depth 1
```

# Клонирование с диска

```
$ git clone <path/to/repo>
```

# Практика

1. С помощью команды `time command>` замерьте сколько времени требуется на клонирование репозитория.

```
$ time git clone https://github.com/libgit2/libgit2 tmp_full
$ time git clone https://github.com/libgit2/libgit2 --depth 1 tmp_depth1
$ time git clone tmp_full tmp_local
$ time git clone tmp_full --depth 1 tmp_local
```

2. Очистите диск:

```
$ rm -rf tmp_full tmp_depth1 tmp_local tmp_local
```

# Импорт репозитория из Subversion

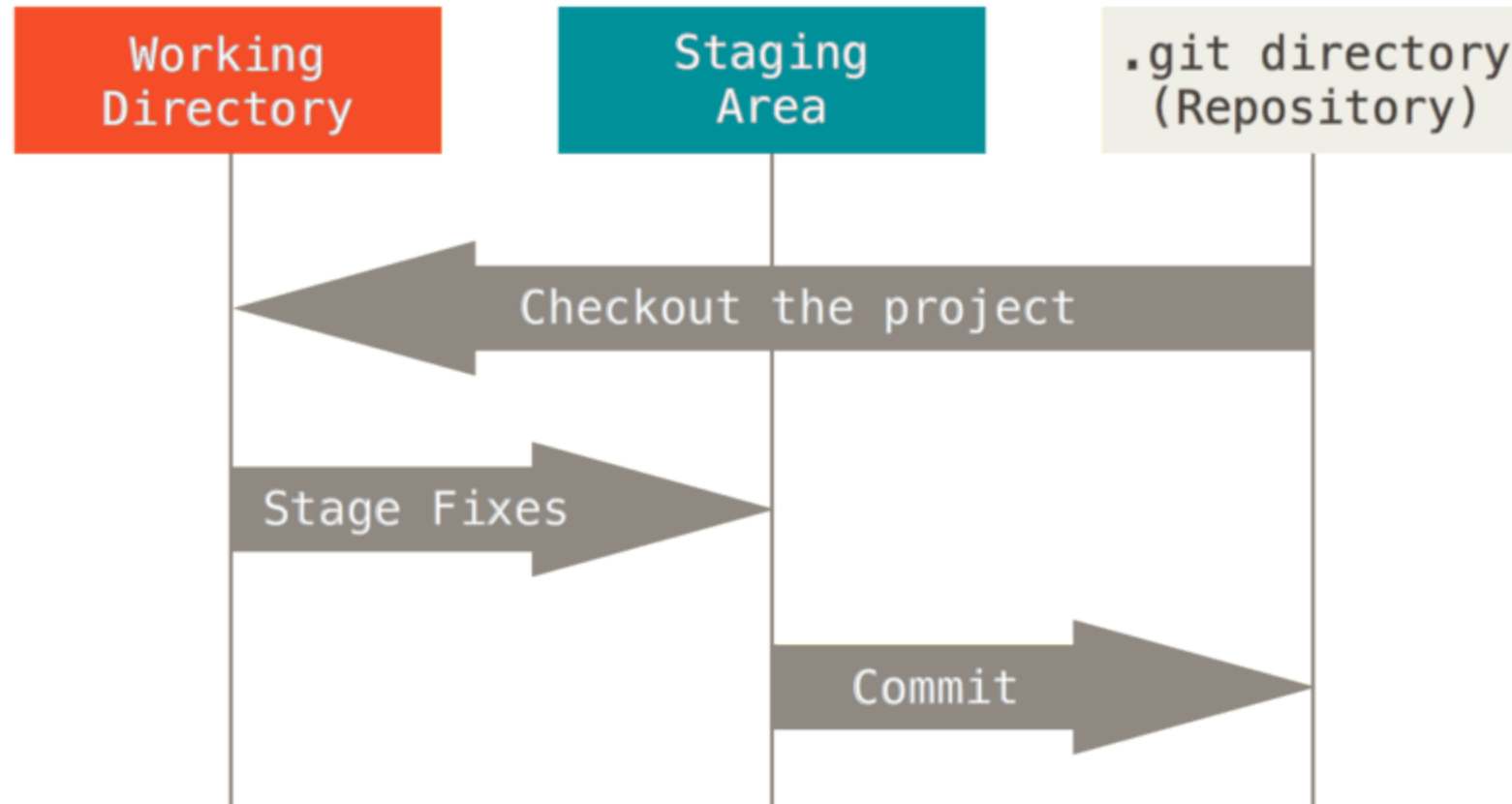
Git поставляется со скриптом `git-svn` который имеет команду клонирования которая импортирует репозиторий subversion в новый git репозиторий. Также существует бесплатная утилита на GitHub которая может это сделать.

```
$ git-svn clone http://projects.com/svn/trunk new-project
```

Это даст вам новый Git репозиторий со всей историей оригинального репозитория Subversion. Это занимает большое количество времени, обычно она начинается с версии 1 и извлекает и выполняет коммиты локально на каждый один снимок.



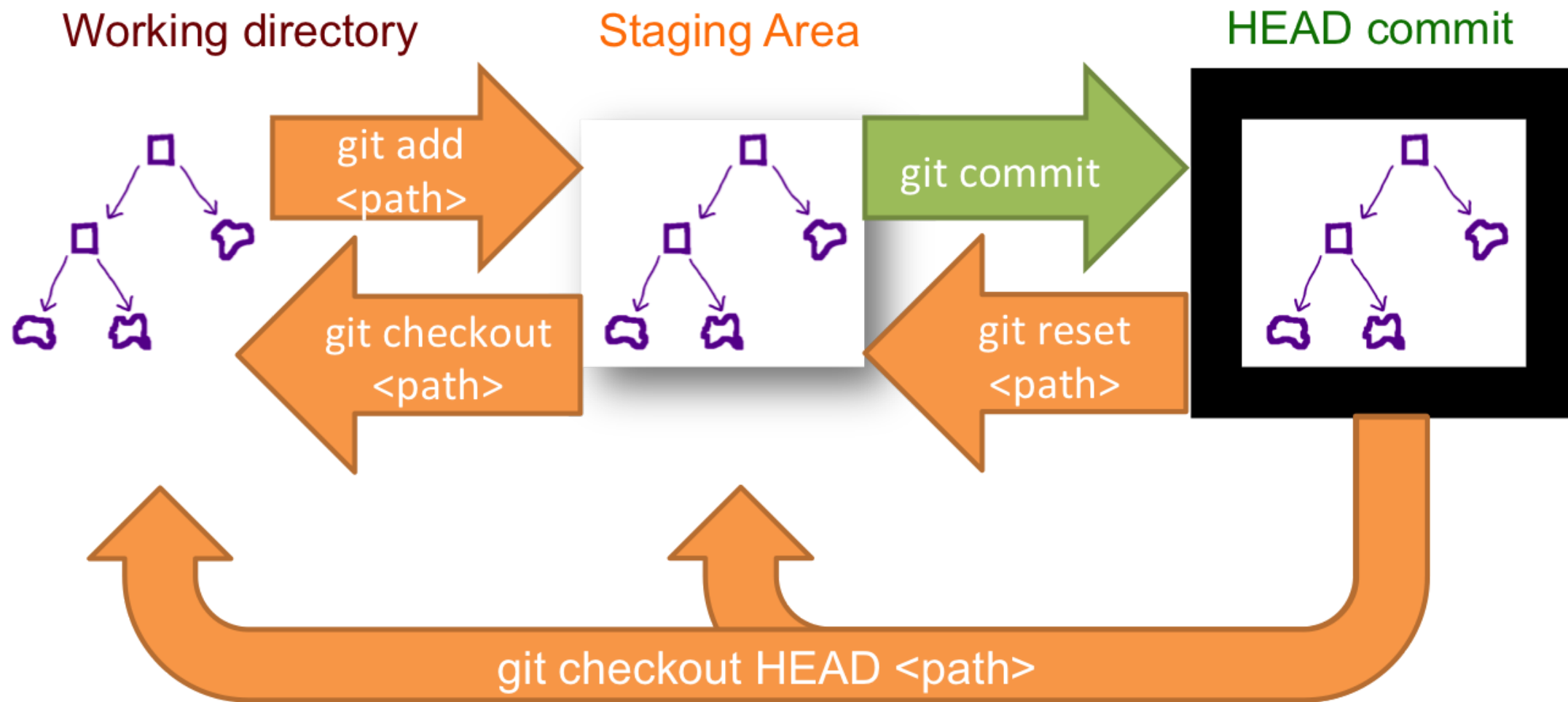
# Области хранения файлов



## Области хранения файлов (продолжение)

- **Git-директория** - место, где Git хранит метаданные и базу объектов вашего проекта. Это самая важная часть Git, и это та часть, которая копируется при клонировании репозитория с другого компьютера.
- **Рабочая директория** является снимком версии проекта. Файлы распаковываются из сжатой базы данных в Git-директории и располагаются на диске, для того чтобы их можно было изменять и использовать.
- **Область подготовленных файлов (индекс)** - это файл, обычно располагающийся в вашей Git-директории, в нём содержится информация о том, какие изменения попадут в следующий коммит.

## Области хранения файлов (продолжение)



# Базовый подход в работе с Git

1. Вы *изменяете* файлы в вашей рабочей директории.
2. Вы выборочно *добавляете* в индекс только те изменения, которые должны попасть в следующий коммит, добавляя тем самым снимки только этих изменений в область подготовленных файлов.
3. Когда вы *делаете коммит*, используются файлы из индекса как есть, и этот снимок сохраняется в вашу Git-директорию.

# Получение состояния файлов

Команда `git status` показывает состояние файлов.

Вывод для чистого репозитория:

```
$ git status
On branch master
No commits yet
nothing to commit (create/copy files and use "git add" to track)
```

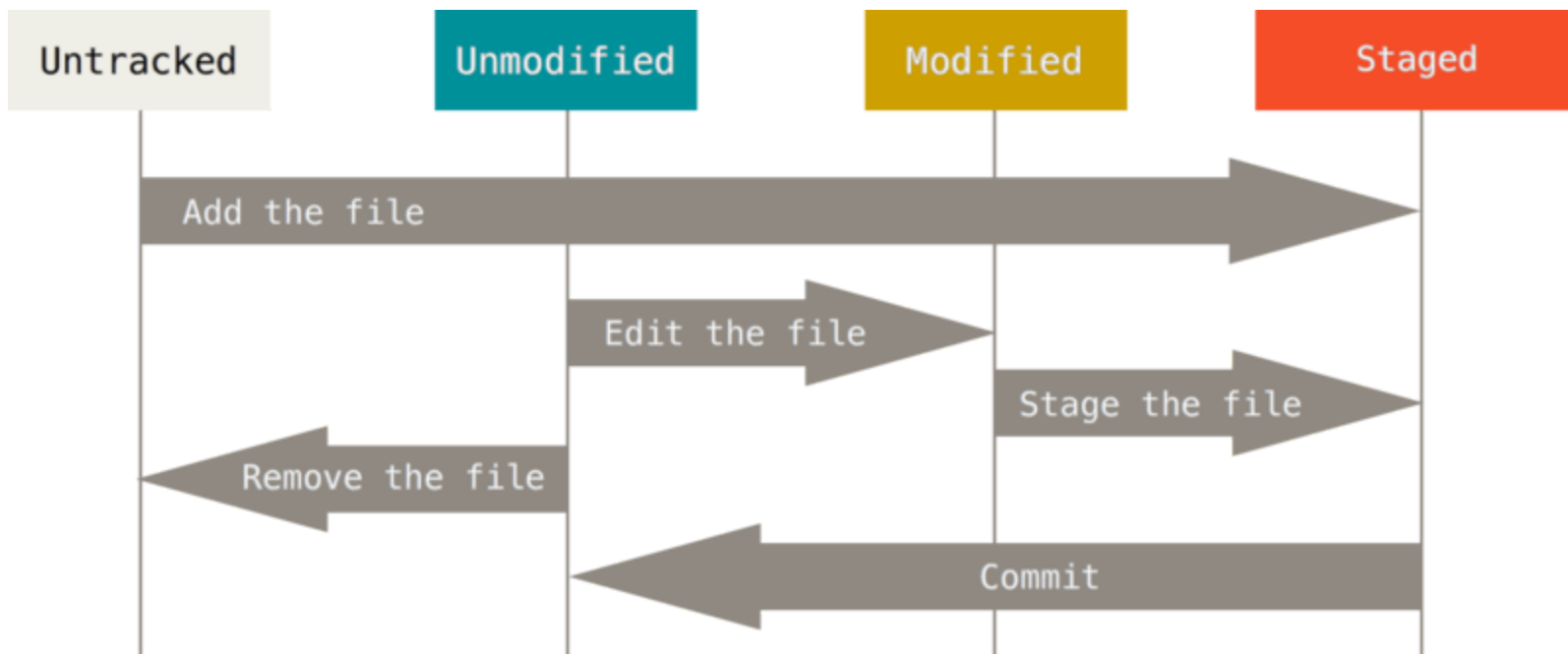
Вывод при наличии неотслеживаемых (новых) файлов:

```
$ echo "My Project" > README
$ git status
On branch master
No commits yet
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)
```

# Состояния файла

- **неотслеживаемые** (untracked) - файлы, которые не были добавлены под версионный контроль
- **отслеживаемые** (tracked) - файлы, которые находятся под версионным контролем:
  - **изменённые** (modified) - файлы, которые находятся под версионным контролем и поменялись, но ещё не были зафиксированы
  - **подготовленные** (staged) - изменённые файлы, отмеченные для включения в следующий коммит
  - **зафиксированные** (unmodified/committed) - файлы сохранённые в вашей локальной базе

## Состояния файла (продолжение)



# Добавление изменений

Команда `git add <file>` добавляет под версионный контроль новые файлы и обновляет существующие:

```
$ git add README
$ git status
On branch master
No commits yet
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   README
```

👉 Можно добавлять сразу несколько файлов и даже папки (часто встречается `.`):

```
$ git add <file1> <file2> <folder1> <folder2> ...
```



# Добавление изменений (продолжение)

Копия добавленных изменений хранится в индексе:

```
$ echo "v0.1" >> README
$ git status
On branch master
No commits yet
Changes to be committed:
  (use "git rm --cached <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:   README
```

👉 Изменения на диске автоматически *не* попадают в коммит (даже если файл был ранее добавлен через `git add`).

# Удаление файлов

Команда `git rm` удаляет файл на диске и из-под версионного контроля:

```
git rm [-r] <file> ...
```

Пример:

```
$ git rm README.txt
rm 'README.txt'

$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:    README.txt
```

## Удаление файлов (продолжение)

Для удаления папок дополнительно укажите флаг `-r`:

```
$ git rm Reports/  
fatal: not removing 'Reports/' recursively without -r  
  
$ git rm -r Reports/  
rm 'Reports/20210408.txt'
```

# Переименование файлов

Переименование файлов осуществляется командой `git mv`:

```
git mv <source> <destination>
```

Пример:

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

$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    renamed:   README.txt -> README.md
```

# Запись изменений в репозиторий

Команда `git commit` создаёт коммит из файлов, помещённых в индекс:

```
$ git commit  
$ git commit -m "<сообщение>"
```

В первом примере откроется редактор, согласно настройке `core.editor`.

Во втором примере текст сообщения коммита берётся из командной строки.

## Запись изменений в репозиторий (продолжение)

```
$ git commit -m "Add README"
[master 463dc4f] Add README
 2 files changed, 2 insertions(+)
 create mode 100644 README
```

Если использовать текстовый редактор, то Git заполняет сообщение по определённому шаблону:

```
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
# Changes to be committed:
#   new file:   README
```

# Полезности команды `git add` 😊

Опция `-A` позволяет сразу

- добавить в индекс новые файлы (которые *не* находятся под версионным контролем)
- добавить в индекс изменённые файлы (которые находятся под версионным контролем)
- удалить из индекса файлы, которые были удалены в рабочей папке

Опция `-n` или `--dry-run` позволяет симитировать добавление файлов. Вы получаете вывод команды, как будто она была выполнена.

- 🙌 это опция есть у многих команд Git

# Практика

Выполните следующие шаги. После каждого шага проверяйте статус файлов ( `git status` ):

1. Перейти в папку с новым репозиторием (команда `cd` )
2. Добавьте несколько файлов (команда `git add` )
3. Создайте первый коммит ( `git commit` )
4. Добавьте ещё несколько файлов (команда `git add` )
5. Создайте второй коммит ( `git commit -m "<сообщение>"` )
6. Переименуйте один файл, а другой удалите
7. Создайте третий коммит ( `git commit` ). Обратите на изменение комментариев в сообщении коммита.



# Просмотр изменений

Команда `git status` показывает изменения в самом общем виде, перечисляя имена файлов.

Команда `git diff` показывает вам непосредственно добавленные и удалённые строки - собственно заплатку (patch).

# Просмотр неиндексированных изменений

Просмотр неиндексированных (до `git add`) изменений:

```
$ git diff
diff --git a/README b/README
index 56266d3..0a94255 100644
--- a/README
+++ b/README
@@ -1,2 @@
  My Project
+v0.1
```

# Просмотр индексируемых изменений

Просмотр индексируемых (после `git add`) изменений:

```
$ git diff --cached
diff --git a/README b/README
new file mode 100644
index 0000000..56266d3
--- /dev/null
+++ b/README
@@ -0,0 +1 @@
+My Project
```

# Просмотр изменений во внешних программах

Команда `git difftool` (вместо `git diff`) позволяет использовать внешние программы для просмотра изменений.

Просмотр доступных программ (вывод урезан):

```
$ git difftool --tool-help
'git difftool --tool=<tool>' may be set to one of the following:
  meld
  p4merge
  vimdiff
user-defined:
  bc.cmd "C:/Program Files/Beyond Compare 4/bcomp.exe" "$LOCAL" "$REMOTE"
  bc.cmd "C:/Program Files/Beyond Compare 4/bcomp.exe" "$LOCAL" "$REMOTE" "$BASE" "$MERGED"
The following tools are valid, but not currently available:
  araxis
  codecompare
  ...
  xxdiff
```

Some of the tools listed above only work in a windowed environment. If run in a terminal-only session, they will fail.

# Просмотр изменений во внешних программах (продолжение)

Запуск внешней программы сравнения:

```
$ git difftool [-t название] [путь к файлу]
$ git difftool
$ git difftool -t p4merge
$ git difftool -t p4merge README.txt
```

Добавить свою программу можно, например, так (если её нет в списке `git difftool --tool-help`):

```
$ git config --global diff.tool p4merge
$ git config --global difftool.p4merge.path \
  "C:/Program Files/Perforce/p4merge.exe"
$ git config --global difftool.p4merge.cmd \
  "'C:/Program Files/Perforce/p4merge.exe' %LOCAL% %REMOTE%"
$ git config --global difftool.prompt false
```

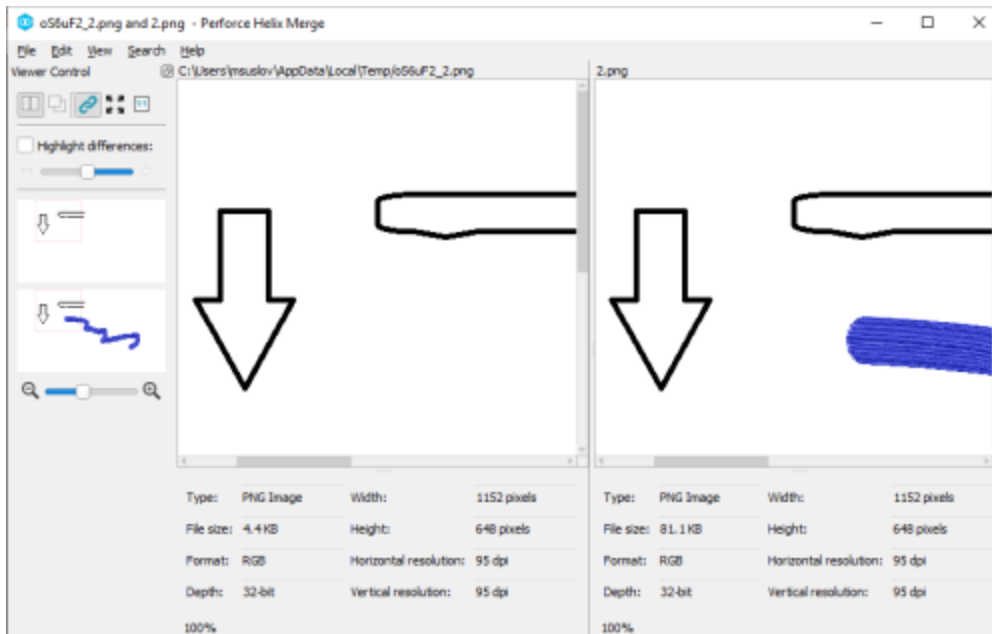
# Сравнение бинарных файлов

Git не показывает различия в бинарных файлах (только сам факт различия):

- использовать внешние программы сравнения (многие из них позволяют сравнивать, например, изображения)
- преобразовать содержимое файлов в текстовый вид

# Сравнение бинарных файлов → Изображения

Например, r4merge может сравнивать не только текстовые, но и графические файлы:



BeyondCompare может сравнивать даже содержимое архивов (и много чего ещё).

# Сравнение бинарных файлов → Преобразование данных

Чтобы сравнить бинарные данные можно преобразовать их в текстовый формат, который понимает Git. Например:

- PDF-документы можно преобразовать в HTML
- архивы заменить списком файлов в них
- для графических файлов сравнивать метаданные (размеры, атрибуты и другие данные изображения)

```
$ git config diff.pdfconv.textconv "pdftohtml -stdout"
$ cat .gitattributes
*.pdf diff=pdfconv
$ git diff
<HTML>
<BODY>
Interior. Room filled with monkeys bashing on typewriters. Mr. Burns tears a sheet of paper from a typewriter.<br>
-Burns: It was the best of times.. it was the blurst of times?<br>
+Burns: It was the best of times.. it was the blurst of times? You stupid monkey!<br>
Burns balls up the sheet and throws it at the monkey.<br>
</BODY>
</HTML>
```



# Практика

1. Сделать изменения в файлах репозитория.
2. Посмотреть изменения, используя `git diff`.
  - насколько всё понятно? есть ли вопросы по формату?
3. Посмотреть перечень доступных программ сравнения `git difftool --tool-help`
4. Посмотреть изменения, используя `git difftool`.
  - попробуйте несколько вариантов, какой подошёл лучше?

# Заключение

Команда	Назначение
<code>git config</code>	Настройка Git
<code>git help</code>	Получение справки по команде
<code>git init</code>	Создание нового репозитория
<code>git clone</code>	Создание копии репозитория

## Заключение (продолжение)

Команда	Назначение
<code>git add</code>	Добавить файлы
<code>git rm</code>	Удалить файлы
<code>git mv</code>	Переименовать файлы
<code>git diff</code>	Просмотр изменений
<code>git commit</code>	Сохранение изменений

**Ваши вопросы ?**