

codereviewCVS
mercurial
git
svn
bzip
arch



V4.1_29.06.217



V2.1_26.06.217



V3.1_28.06.217



V1.2_25.06.217



V2.2_27.06.217



V1.1_25.06.217

Source Code Control System

Source Code Control System (SCCS) — первая система управления версиями, разработанная в Bell Labs в 1972 году Марком Рочкиндо (англ. Marc J. Rochkind) для компьютеров IBM System/370.

В дальнейшем была создана версия для PDP-11 под управлением операционной системы UNIX. В дальнейшем SCCS была включена в состав нескольких вариантов UNIX. Набор команд SCCS в настоящее время является частью Single UNIX Specification.

SCCS являлась самой распространённой системой управления версиями до появления RCS. Несмотря на то, что в настоящее время SCCS следует признать устаревшей системой, формат файлов, разработанный для SCCS, до сих пор используется некоторыми системами управления версиями, такими как BitKeeper и TeamWare. Для хранения изменений SCCS использует т. н. технику чередующихся изменений (англ. interleaved deltas). Данная техника используется многими современными системами управления версиями в качестве основы для изолированных методов слияния.

Revision Control System

RCS (Revision Control System) является одной из самых первых систем управления версиями.

Для каждого файла, зарегистрированного в системе, она **хранит полную** историю изменений, причём для текстовых файлов используется эффективный алгоритм дельта-компрессии (<http://qps.ru/fvaGB>), когда хранится только последняя версия и все межверсионные изменения.

Система позволяет также хранить версии бинарных файлов, но без использования дельта-компрессии, то есть каждая версия бинарного файла хранится полностью.



Система контроля версий — это система, записывающая изменения в файл или набор файлов в течение времени и позволяющая вернуться позже к определённой версии.

Система контроля версий позволяет вернуть файлы к состоянию, в котором они были до изменений, вернуть проект к исходному состоянию, увидеть изменения, увидеть, кто последний менял что-то и вызвал проблему, кто поставил задачу и когда, и многое другое.

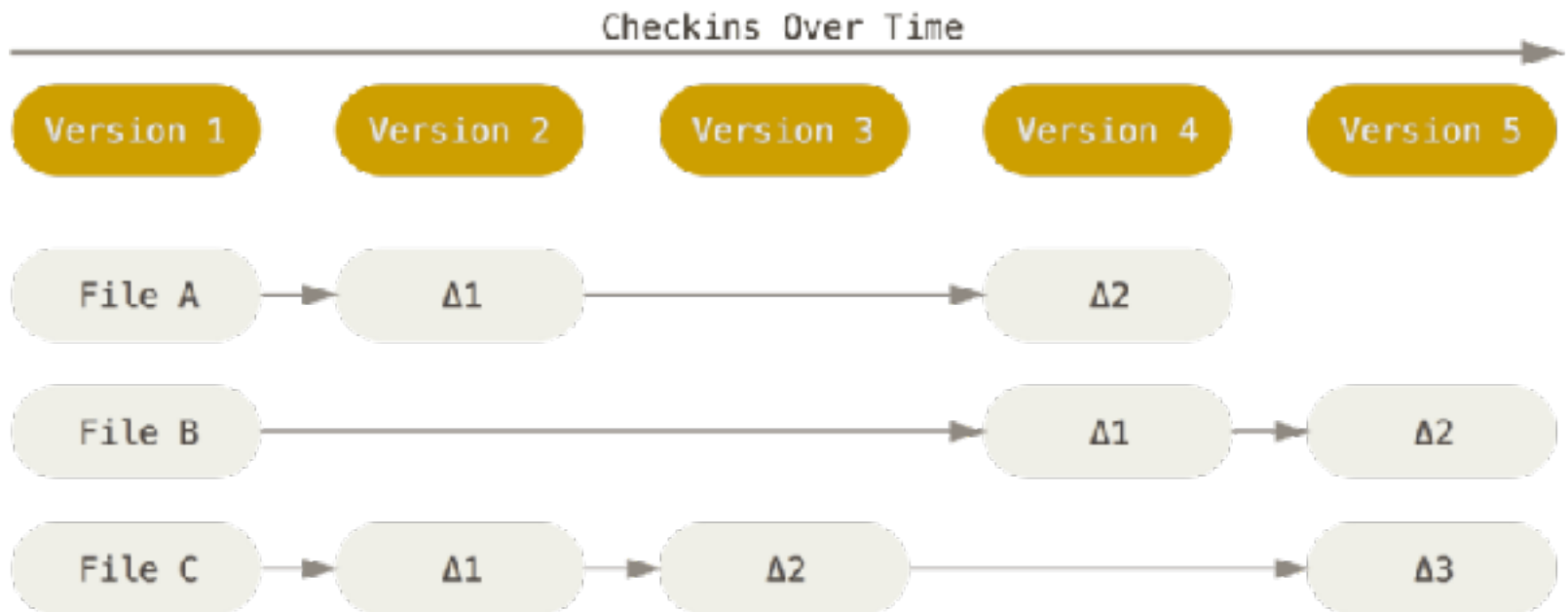
Использование системы контроля версий покажет, что если вы сломали что-то или потеряли файлы, вы спокойно можете всё исправить.

Можно использовать контроль версий для любых типов файлов



Git хранит и использует информацию совсем иначе по сравнению с другими системами, даже несмотря на то, что интерфейс пользователя достаточно похож.

Основное отличие Git'a от любой другой системы - это подход Git'a к работе со своими данными. Концептуально, большинство других систем хранят информацию в виде списка изменений в файлах. Эти системы представляют информацию в виде набора файлов и изменений, сделанных в каждом файле, по времени.

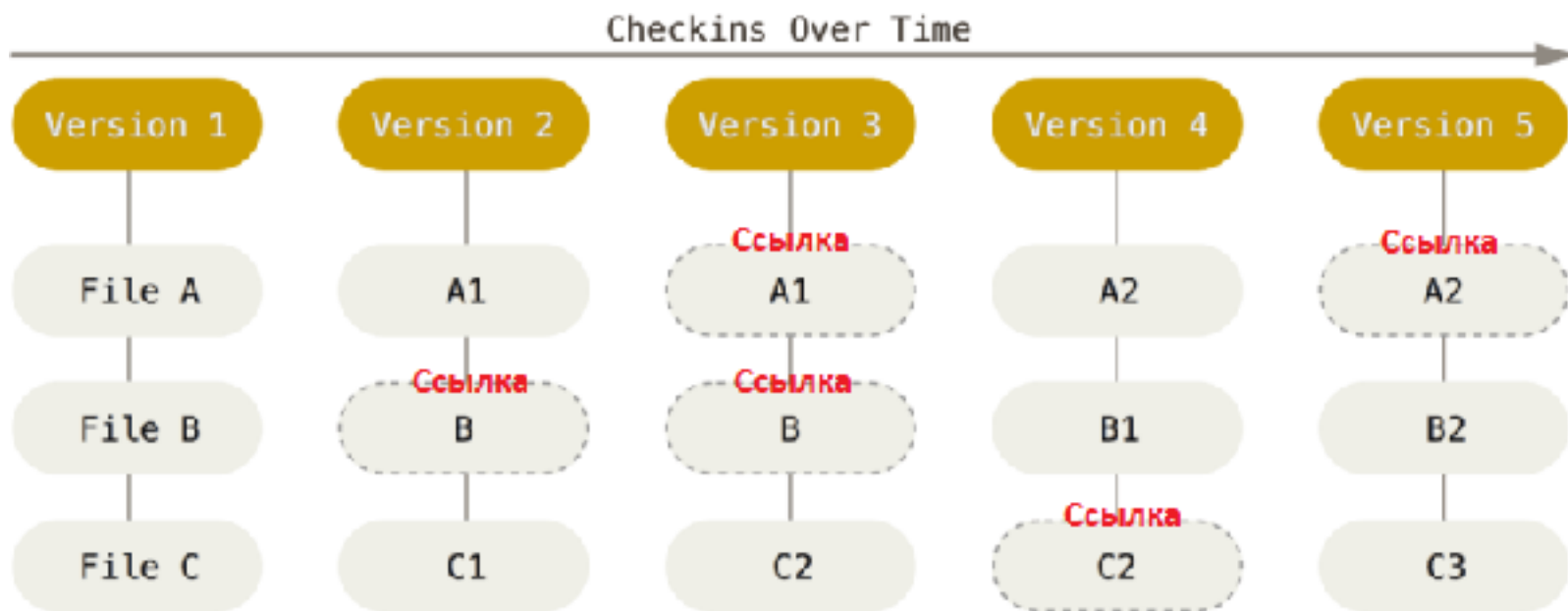


Git не хранит и не обрабатывает данные таким способом.

Вместо этого, подход Git'a к хранению данных больше похож на набор снимков миниатюрной файловой системы.

Каждый раз, когда вы делаете коммит, то есть сохраняете состояние своего проекта в Git'e, система запоминает, как выглядит каждый файл в этот момент, и сохраняет ссылку на этот снимок.

Для увеличения эффективности, если файлы не были изменены, Git не запоминает эти файлы вновь, а только создаёт ссылку на предыдущую версию идентичного файла, который уже сохранён. Git представляет свои данные как, скажем, **ПОТОК СНИМКОВ**.



Целостность Git

В Git'е для всего вычисляется хеш-сумма, и только потом происходит сохранение.

В дальнейшем обращение к сохранённым объектам происходит по этой хеш-сумме.

Это значит, что невозможно изменить содержимое файла или директории так, чтобы Git не узнал об этом.

Данная функциональность встроена в Git на низком уровне.

Вы не потеряете информацию во время её передачи и не получите повреждённый файл без ведома Git.

Механизм, которым пользуется Git при вычислении хеш-сумм, называется SHA-1 хеш. Это строка длиной в 40 шестнадцатеричных символов (0-9 и a-f), она вычисляется на основе содержимого файла или структуры каталога. SHA-1 хеш выглядит примерно так:

24b9da6552252987aa493b52f8696cd6d3b00373

Вы будете постоянно встречать хеши в Git'е, потому что он использует их повсеместно. На самом деле, Git сохраняет все объекты в свою базу данных не по имени, а по хеш-сумме содержимого объекта.

Состояния файлов git

Git имеет **три основных состояния**, в которых могут находиться ваши файлы:
зафиксированном (committed),
изменённом (modified) и
подготовленном (staged).

Зафиксированный —
файл уже сохранён в
вашей локальной базе.
git commit

```
jo@jo-XPS-13-9350:~/gitrepositories/foss4gukdontbeafraid$ git commit
[master (root-commit) 45cf2eb] Created repository and first file (cur
nk)
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 blah.txt
jo@jo-XPS-13-9350:~/gitrepositories/foss4gukdontbeafraid$ git status
On branch master
nothing to commit, working directory clean
jo@jo-XPS-13-9350:~/gitrepositories/foss4gukdontbeafraid$
```

К изменённым относятся
файлы, которые поменялись,
но ещё не были зафиксированы.
git status

```
ukky@DESKTOP-SKJLS2K MINGW64 ~/git/repos/_test_/git test (dev)
$ git status
On branch dev
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:   divide.js

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

Подготовленные файлы — это
изменённые файлы, отмеченные
для включения в следующий коммит.
git add .

```
jo@jo-XPS-13-9350:~/gitrepositories/foss4gukdontbeafraid$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   blah.txt

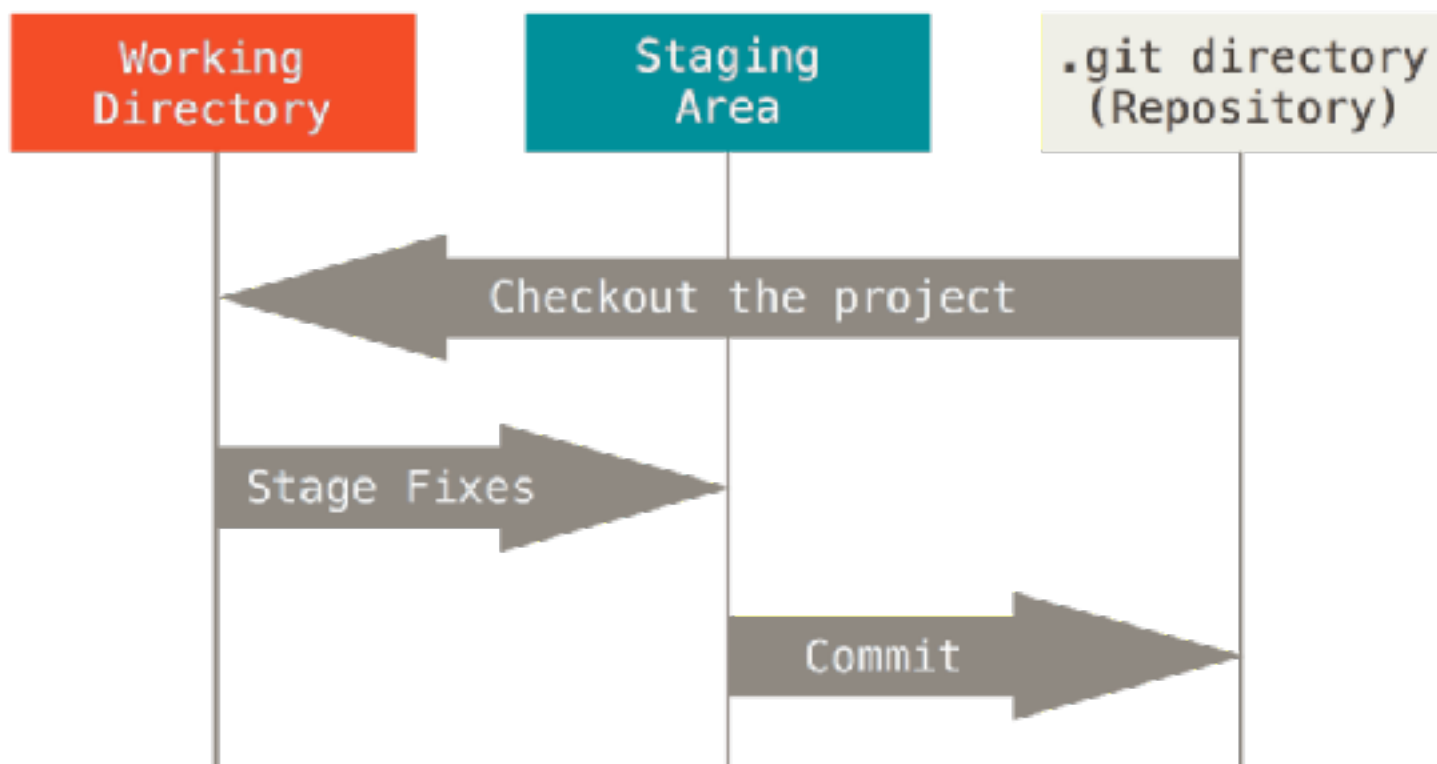
jo@jo-XPS-13-9350:~/gitrepositories/foss4gukdontbeafraid$
```

Три основные секции проекта Git:

Git-директория (Git directory),

рабочая директория (working directory) и

область подготовленных файлов (staging area).



Git-директория — это то 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 — это файл, располагающийся в вашей Git-директории, в нём содержится информация о том, какие изменения попадут в следующий коммит. Эту область ещё называют “индекс”, однако называть её stage-область также общепринято.

Базовый подход в работе с Git выглядит так:

Вы изменяете файлы в вашей рабочей директории.

Вы добавляете файлы в индекс, добавляя тем самым их снимки в область подготовленных файлов.

Когда вы делаете коммит, используются файлы из индекса как есть, и этот снимок сохраняется в вашу Git директорию.

Установка git

Downloading Git



Your download is starting...

You are downloading the latest (2.19.0) 64-bit version of Git for Windows. This is the most recent [maintained build](#). It was released 18 days ago, on 2018-09-13.

If your download hasn't started, [click here to download manually](#).

Other Git for Windows downloads

Git for Windows Setup

[32-bit Git for Windows Setup](#).

[64-bit Git for Windows Setup](#).

Git for Windows Portable ("thumbdrive edition")

[32-bit Git for Windows Portable](#).

[64-bit Git for Windows Portable](#).

The current source code release is version 2.19.0. If you want the newer version, you can build it from [the source code](#).

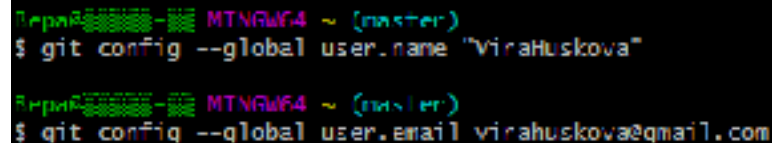
```
Bepa@MINGW64 ~  
$ git init  
Initialized empty Git repository in C:/Users/Bepa/.git/
```

Необходимо настроить среду для работы с Git'ом. Это нужно сделать только один раз — при обновлении версии Git'a настройки сохранятся.

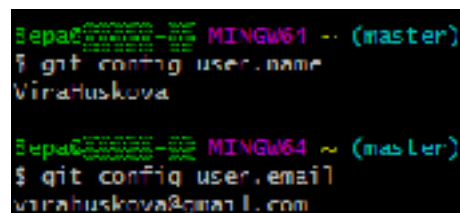
В состав Git'a входит утилита `git config`, которая позволяет просматривать и настраивать параметры, контролирующие все аспекты работы Git'a, а также его внешний вид.

Первое, что вам следует сделать после установки Git'a, — указать ваше имя и адрес электронной почты. Это важно, потому что каждый коммит в Git'e содержит эту информацию, и она включена в коммиты, передаваемые вами, и не может быть далее изменена.

```
$ git config --global user.name « »  
$ git config --global user.email
```



```
Бера8888888888888888 MINGW64 ~ (master)  
$ git config --global user.name "Virahuskova"  
  
Бера8888888888888888 MINGW64 ~ (master)  
$ git config --global user.email virahuskova@gmail.com
```



```
Бера8888888888888888 MINGW64 ~ (master)  
$ git config user.name  
Virahuskova  
  
Бера8888888888888888 MINGW64 ~ (master)  
$ git config user.email  
virahuskova@gmail.com
```

Редактор по умолчанию Vim

Для того, что бы изменить редактор (для Emacs)

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

Проверить используемую конфигурацию можно с `git config --list`

```
Вера@DESKTOP-080838-MINGW64 ~ (master)
$ git config --list
core.symlinks=false
core.autocrlf=true
core.fscache=true
color.diff=auto
color.status=auto
color.branch=auto
color.interactive=true
help.format=html
diff.astextplain.textconv=astextplain
rebase.autosquash=true
http.sslcainfo=C:/Program Files/Git/mingw64/ssl/certs/ca-bundle.crt
http.sslbackend=openssl
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
credential.helper=manager
rebase.usebuiltin=true
stash.usebuiltin=true
```

\$ git help

```
Bepa@MINGW64 ~ (master)
```

```
$ git help
```

```
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
      [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
      [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
      [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
      <command> [<args>]
```

These are common Git commands used in various situations:

start a working area (see also: `git help tutorial`)

<code>clone</code>	Clone a repository into a new directory
<code>init</code>	Create an empty Git repository or reinitialize an existing one

work on the current change (see also: `git help everyday`)

<code>add</code>	Add file contents to the index
<code>mv</code>	Move or rename a file, a directory, or a symlink
<code>reset</code>	Reset current HEAD to the specified state
<code>rm</code>	Remove files from the working tree and from the index

examine the history and state (see also: `git help revisions`)

<code>bisect</code>	Use binary search to find the commit that introduced a bug
<code>grep</code>	Print lines matching a pattern
<code>log</code>	Show commit logs
<code>show</code>	Show various types of objects
<code>status</code>	Show the working tree status

Создание git репозитория

Создание репозитория в существующей директории

Использование git для существующего проекта. Перейти в папку проекта и ввести

```
$ git init
```

Команда создаёт в текущей директории новую поддиректорию с именем `.git`, содержащую все необходимые файлы репозитория — основу Git-репозитория.

Добавить под версионный контроль существующие файлы - добавить их в индекс и осуществить первый коммит изменений.

Запустив команду `git add`, указав индексируемые файлы, а затем выполнив `git commit`:

```
$ git add *.c  
$ git add LICENSE  
$ git commit -m 'initial project version'
```

Клонирование существующего репозитория

Для получения копии существующего Git-репозитория, например, проекта, в который вы хотите внести свой вклад, необходимо использовать команду `git clone`.

При выполнении `git clone` с сервера забирается (pulled) каждая версия каждого файла из истории проекта. Фактически, если серверный диск выйдет из строя, вы можете использовать любой из клонов на любом из клиентов, для того, чтобы вернуть сервер в то состояние, в котором он находился в момент клонирования (вы можете потерять часть серверных перехватчиков (server-side hooks) и т.п., но все данные, помещённые под версионный контроль, будут сохранены).

Клонирование репозитория осуществляется командой `git clone [url]`.

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

Эта команда создаёт директорию, инициализирует в ней поддиректорию `.git`, скачивает все данные для этого репозитория и создаёт рабочую копию последней версии.

Если зайти в новую директорию, то в ней есть файлы проекта, готовые для работы или использования.

В Git'e реализовано несколько транспортных протоколов. `https://` и `git://` или `user@server:path/to/repo.git`, использующий протокол передачи SSH.

Clone this repository

SSH ▾

HTTPS

git clone git@bitbucket.org:

[Sourcetree](#) is a free Git and Mercurial client for macOS.

Clone in Sourcetree

Close

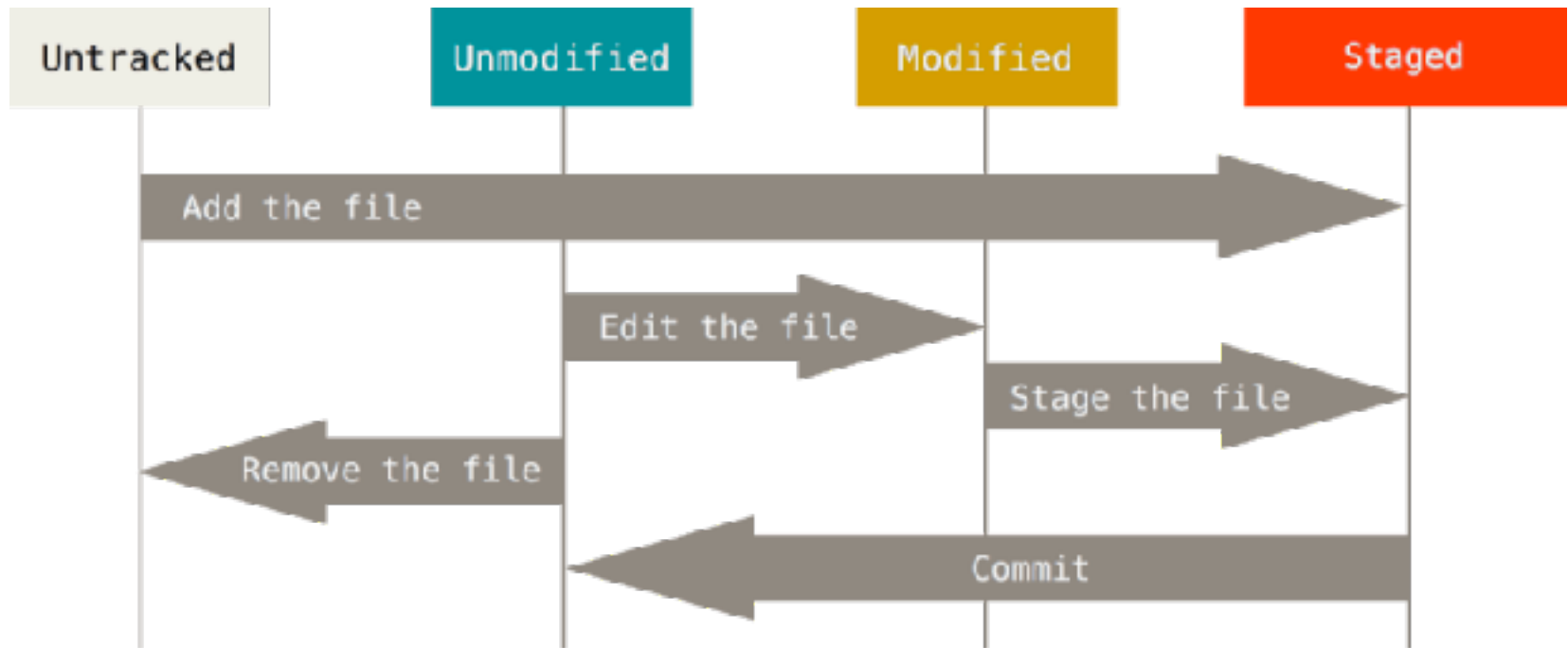
Запись изменений в репозиторий

Каждый файл в рабочем каталоге может находиться в одном из двух состояний: под версионным контролем (отслеживаемые) и нет (неотслеживаемые).

Отслеживаемые файлы — это те файлы, которые были в последнем слепке состояния проекта (snapshot); они могут быть неизменёнными, изменёнными или подготовленными к коммиту (staged).

Неотслеживаемые файлы — это всё остальное, любые файлы в вашем рабочем каталоге, которые не входили в ваш последний слепок состояния и не подготовлены к коммиту. Когда вы впервые клонируете репозиторий, все файлы будут отслеживаемыми и неизменёнными, потому что вы только взяли их из хранилища (checked them out) и ничего пока не редактировали.

Как только вы отредактируете файлы, Git будет рассматривать их как изменённые, т.к. вы изменили их с момента последнего коммита. Вы индексируете эти изменения и затем фиксируете все индексированные изменения, а затем цикл повторяется.



Состояние файлов

```
$ git status  
On branch master  
nothing to commit, working directory clean
```

Команда сообщает:

1. Неотслеживаемые файлы
2. Изменения в каталоге
3. Ветку (на которой вы находитесь)

Добавление новых файлов на отслеживание

Чтобы начать отслеживать (добавить под версионный контроль) новый файл, используется команда `git add`.

```
$ git add README
```

`git status`, отобразится, что файл README теперь отслеживаемый и индексированный

```
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   README
```

Команда `git add` принимает параметром путь к файлу или каталогу, если это каталог, команда рекурсивно добавляет (индексирует) все файлы в данном каталоге.

Индексация изменений

Если модифицировать файл, находящийся под версионным контролем и написать `git status`, то результат будет следующим:

```
→ test git:(master) ✕ git status
On branch 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:   main.html

no changes added to commit (use "git add" and/or "git commit -a")
→ test git:(master) ✕
```

```
→ test git:(master) ✕ git status
On branch 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:   main.html

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

        index.html

no changes added to commit (use "git add" and/or "git commit -a")
→ test git:(master) ✕
```

Git индексирует файл в точности в том состоянии, в котором он находился, когда вы выполнили команду `git add`. Если вы изменили файл после выполнения `git add`, вам придётся снова выполнить `git add`, чтобы проиндексировать последнюю версию файла:

```
→ test git:(master) ✕ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   main.html

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:   main.html

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

        index.html

→ test git:(master) ✕
```

Вывод статуса

Git также имеет флаг вывода сокращенного статуса, так что вы можете увидеть изменения в более компактном виде. Если выполнить `git status -s` или `git status --short` получим упрощенный вывод.

```
→ test git:(master) x git status -s
MM main.html
?? index.html
→ test git:(master) x
```

?? неотслеживаемые файлы,

M файлы добавленные в отслеживаемые,

M отредактированные файлы

Игнорируемые файлы



Имеется группа файлов, которые вы не только не хотите автоматически добавлять в репозиторий, но и видеть в списках неотслеживаемых.

К таким файлам обычно относятся автоматически генерируемые файлы (различные логи, результаты сборки программ и т.п.).

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

```
$ cat .gitignore
*.oa
*~
```

```
6  .Ds_Store
7  *~
8  *.egg-info
9  *.pot
10 *.py[co]
11 *.mo
12 *.gz
13 *.zip
14 *.dat
15 *.log
16 *.sh
17 favicon.ico
18 favicon*.png
19 logo.png
20 html/images/slides/sl_*
21 settings/local.py
22 bower_components/
23 node_modules/
24 public/media/
25 public/static/
26 build
```

К шаблонам в файле `.gitignore` применяются следующие правила:

- Пустые строки, а также строки, начинающиеся с `#`, игнорируются.
- Можно использовать стандартные glob шаблоны (упрощённые регулярные выражения).
- Можно начать шаблон символом слэша (/) чтобы избежать рекурсии.
- Можно заканчивать шаблон символом слэша (/) для указания каталога.
- Можно инвертировать шаблон, использовав восклицательный знак (!) в качестве первого символа.

Просмотр индексированных и неиндексированных изменений

```
git status
```

```
git diff
```

```
→ test git:(master) ✕ git status
On branch master
Changes to be committed:
  diff --git a/main.html b/main.html
      index 17fa266..c75df2b 100644
      --- a/main.html
      +++ b/main.html
      @@ -12,5 +12,6 @@
           <h5>Hello</h5>
           <h6>Hello</h6>
           <span>text</span>
      +   <p>text1</p>
      </body>
      </html>
      (END)
```

Что уже проиндексировали и что войдёт в следующий коммит - `git diff --staged`. Эта команда сравнивает ваши индексированные изменения с последним коммитом:

```
diff --git a/main.html b/main.html
index 49c8407..17fa266 100644
--- a/main.html
+++ b/main.html
@@ -11,5 +11,6 @@
     <h4>Hello</h4>
     <h5>Hello</h5>
     <h6>Hello</h6>
+   <span>text</span>
   </body>
- </html>
\ No newline at end of file
+</html>
```

`git diff` не показывает все изменения сделанные с последнего коммита — только те, что ещё не проиндексированы

Коммит изменений

Всё, что не проиндексировано — любые файлы, созданные или изменённые, и для которых не выполнили `git add` после момента редактирования — не войдут в этот коммит.

Фиксация изменений `git commit` - вызывает редактор для добавления комментария

```
$ git commit
```

Добавить комментарий без открытия редактора - добавление флага `-m` «текст сообщения»

```
$ git commit -m « »
```

```
→ test git:(master) ✕ git commit -m "Added test files"
[master 796da37] Added test files
2 files changed, 4 insertions(+), 1 deletion(-)
create mode 100644 index.html
```

Коммит вывел информации: на какую ветку выполнили коммит (`master`), какая контрольная сумма SHA-1 у этого коммита (`463dc4f`), сколько файлов было изменено, а также статистику по добавленным/удалённым строкам в этом коммите.

Удаление файлов

Чтобы удалить файл из Git, необходимо удалить его из отслеживаемых файлов, а затем выполнить коммит.

```
git rm
```

```
→ test git:(master) git status
On branch master
nothing to commit, working tree clean
→ test git:(master) git rm main.html
rm 'main.html'
→ test git:(master) x git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        deleted:    main.html

→ test git:(master) x
```

Перемещение файлов

(переименование файлов)

```
$ git mv file_from file_to
```

```
→ test git:(master) ✗ git mv index.html main.html
→ test git:(master) ✗ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        deleted:    index.html
        modified:   main.html

→ test git:(master) ✗
```

История коммитов

```
git log
```

```
→ test git:(master) ✕ git commit -m "Added test files"
[master 796da37] Added test files
 2 files changed, 4 insertions(+), 1 deletion(-)
 create mode 100644 index.html
commit 796da378d8ac27384526a5de29418ff4f23221ce (HEAD -> master)
Author: Vira Huskova <guskovavera2009@gmail.com>
Date:   Tue Oct 2 17:58:55 2018 +0300

    Added test files

commit 717122b399c2194c7862c91eaeb5ace71f262ff2
Author: Vira Huskova <guskovavera2009@gmail.com>
Date:   Tue Oct 2 16:35:34 2018 +0300

    test
```

По умолчанию `git log` перечисляет коммиты, сделанные в репозитории в обратном к хронологическому порядку – последние коммиты находятся сверху.

Опция	Описание
<code>-p</code>	Показывает патч для каждого коммита.
<code>--stat</code>	Показывает статистику измененных файлов для каждого коммита.
<code>--shortstat</code>	Отображает только строку с количеством изменений/вставок/удалений для команды <code>--stat</code> .
<code>--name-only</code>	Показывает список измененных файлов после информации о коммите.
<code>--name-status</code>	Показывает список файлов, которые добавлены/изменены/удалены.
<code>--abbrev-commit</code>	Показывает только несколько символов SHA-1 чек-суммы вместо всех 40.
<code>--relative-date</code>	Отображает дату в относительном формате (например, "2 weeks ago") вместо стандартного формата даты.
<code>--graph</code>	Отображает ASCII граф с ветвлениями и историей слияний.
<code>--pretty</code>	Показывает коммиты в альтернативном формате. Возможные варианты опций: <code>oneline</code> , <code>short</code> , <code>full</code> , <code>fuller</code> и <code>format</code> (с помощью последней опции вы можете указать свой формат).

git log -p

```
commit 796da378d8ac27384526a5de29418ff4f23221ce (HEAD -> master)
Author: Vira Huskova <guskovavera2009@gmail.com>
Date:   Tue Oct 2 17:58:55 2018 +0300

    Added test files

diff --git a/index.html b/index.html
new file mode 100644
index 0000000..44bd344
--- /dev/null
+++ b/index.html
@@ -0,0 +1 @@
+<p class="">Hello <span class="">hi</span></p>
\ No newline at end of file
diff --git a/main.html b/main.html
index 49c8407..c75df2b 100644
--- a/main.html
+++ b/main.html
@@ -11,5 +11,7 @@
     <h4>Hello</h4>
     <h5>Hello</h5>
     <h6>Hello</h6>
+    <span>text</span>
+    <p>text1</p>
     </body>
-</html>
\ No newline at end of file
+</html>
```

git log -p - -stat

```
On branch master
commit 796da378d8ac27384526a5de29418ff4f23221ce (HEAD -> master)
Author: Vira Huskova <guskovavera2009@gmail.com>
Date:   Tue Oct 2 17:58:55 2018 +0300

    Added test files
---
index.html | 1 +
main.html  | 4 +++-
2 files changed, 4 insertions(+), 1 deletion(-)
```

git log - - stat

```
On branch master
Changes to be committed:
commit 796da378d8ac27384526a5de29418ff4f23221ce (HEAD -> master)
Author: Vira Huskova <guskovavera2009@gmail.com>
Date:   Tue Oct 2 17:58:55 2018 +0300

    Added test files

index.html | 1 +
main.html  | 4 +++-
2 files changed, 4 insertions(+), 1 deletion(-)

commit 717122b399c2194c7862c91eae5ace71f262ff2
Author: Vira Huskova <guskovavera2009@gmail.com>
Date:   Tue Oct 2 16:35:34 2018 +0300

    test

main.html | 15 ++++++++
1 file changed, 15 insertions(+)
```

Формат вывода

`git log -pretty`

pretty = oneline выводит информацию в одну строку

```
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

796da378d8ac27384526a5de29418ff4f23221ce (HEAD -> master) Added test files
717122b399c2194c7862c91eaeb5ace71f262ff2 test
```

<code>%H</code>	Хеш коммита
<code>%h</code>	Сокращенный хеш коммита
<code>%T</code>	Хеш дерева
<code>%t</code>	Сокращенный хеш дерева
<code>%P</code>	Хеш родителей
<code>%p</code>	Сокращенный хеш родителей
<code>%an</code>	Имя автора
<code>%ae</code>	Электронная почта автора
<code>%ad</code>	Дата автора (формат даты можно задать опцией <code>--date=option</code>)
<code>%ar</code>	Относительная дата автора
<code>%cn</code>	Имя коммитера
<code>%ce</code>	Электронная почта коммитера
<code>%cd</code>	Дата коммитера
<code>%cr</code>	Относительная дата коммитера
<code>%s</code>	Содержание

git log - -graph

```
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        deleted:    main.html
* commit 796da378d8ac27384526a5de29418ff4f23221ce (HEAD -> master)
| Author: Vira Huskova <guskovavera2009@gmail.com>
| Date:   Tue Oct 2 17:58:55 2018 +0300
|
|     Added test files
|
* commit 717122b399c2194c7862c91eaeb5ace71f262ff2
  Author: Vira Huskova <guskovavera2009@gmail.com>
  Date:   Tue Oct 2 16:35:34 2018 +0300

      test
```

Ограничения вывода

```
$ git log --since=2.weeks
```

```
git log --since="2 years 1 day 3 minutes ago"
```

```
git log --since="2008-01-15"
```

Опция	Описание
<code>-(n)</code>	Показывает только последние n коммитов.
<code>--since</code> , <code>--after</code>	Показывает только те коммиты, которые были сделаны после указанной даты.
<code>--until</code> , <code>--before</code>	Показывает только те коммиты, которые были сделаны до указанной даты.
<code>--author</code>	Показывает только те коммиты, в которых запись author совпадает с указанной строкой.
<code>--committer</code>	Показывает только те коммиты, в которых запись committer совпадает с указанной строкой.
<code>--grep</code>	Показывает только коммиты, сообщение которых содержит указанную строку.
<code>-s</code>	Показывает только коммиты, в которых изменение в коде повлекло за собой добавление или удаление указанной строки.