Отчет к лабе номер 6 по ВРПО

Версия GIT

git version 2.43.0.windows.1

Команда для получения списка команд

git --help

git --help

usage: git [-v | --version] [-h | --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>]

[--config-env=<name>=<envvar>] <command> [<args>]

These are common Git commands used in various situations:

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

clone Clone a repository into a new directory

init Create an empty Git repository or reinitialize an existing one

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

add Add file contents to the index

mv Move or rename a file, a directory, or a symlink

restore Restore working tree files

rm Remove files from the working tree and from the index

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

bisect Use binary search to find the commit that introduced a bug

diff Show changes between commits, commit and working tree, etc

grep Print lines matching a pattern

log Show commit logs

show Show various types of objects

status Show the working tree status

grow, mark and tweak your common history

branch List, create, or delete branches

commit Record changes to the repository

merge Join two or more development histories together

rebase Reapply commits on top of another base tip

reset Reset current HEAD to the specified state

switch Switch branches

tag Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)

fetch Download objects and refs from another repository

pull Fetch from and integrate with another repository or a local branch

push Update remote refs along with associated objects

'git help -a' and 'git help -g' list available subcommands and some

concept guides. See 'git help <command>' or 'git help <concept>'

to read about a specific subcommand or concept.

See 'git help git' for an overview of the system.

СОЗДАЕМ РЕПОЗИТОРИЙ

Git init

Git add file.txt

Появился .git

Git status -убеждаемся,что проиндексировали , Выводим коммиты ,ветки и файлы включенные в проект , или невключенные

git commit -m “Мне лично нравятся красные ... А желтые еще не доработаны”-фиксируем изменение в репозитории командой

Зафиксированное состояние

Git log - посмотреть коммиты

commit 4d886d46c9257083ce6644bf2b04105c64db5cea (HEAD -> master)

Author: RikoCkotletoy <riloprase@gmail.com>

Date: Mon Nov 27 15:01:38 2023 +0300

ADDED file.txt

Git branch MYTESTBRANCH

Git checkout MYTESTBRANCH

Git branch - посмотреть текущую ветку

Merge слияние

1 Система контроля версий (СКВ) - это инструмент для отслеживания изменений в файлах и управления версиями проекта. Она используется для сохранения истории изменений, совместной работы над кодом и восстановления предыдущих версий файлов.

2 Git использует принцип хранения файлов в виде снимков (snapshots). Каждый коммит в Git представляет собой полный снимок всех файлов проекта на определенный момент времени.

3 Отличие Git от других систем контроля версий заключается в его распределенной природе, гибкости и скорости работы. Каждая копия репозитория Git является полноценной копией с историей изменений, что позволяет работать оффлайн и легко совмещать изменения.

4Файлы в Git могут находиться в трех основных состояниях: измененные (modified), проиндексированные (staged) и зафиксированные (committed).

5Индексация файла - это процесс добавления измененных файлов в индекс, чтобы они могли быть включены в следующий коммит. Команда git add используется для индексации файлов.

6Фиксация файла (commit) - это создание коммита, который записывает изменения в репозиторий Git. Команда git commit используется для фиксации файлов.

7Команда git config --list позволяет проверить выбранные настройки Git, такие как имя пользователя, адрес электронной почты и другие.

8Для инициализации репозитория в существующей папке используйте команду git init.

9Чтобы указать файлы, за которыми должна следить система, создайте файл .gitignore в корневой директории репозитория и добавьте в него шаблоны имен файлов или директорий, которые должны быть проигнорированы.

10Изменения фиксируются с помощью команды git commit после индексации файлов.

11Команда git status позволяет определить состояние файлов в репозитории Git, показывая измененные, проиндексированные и неотслеживаемые файлы.

12Файл .gitignore создается для игнорирования определенных файлов и директорий при выполнении операций Git. Он содержит шаблоны имен файлов или директорий, которые не должны быть отслеживаемыми или включенными в коммиты.

13Команда git log используется для просмотра истории коммитов в репозитории Git. Она показывает список коммитов с информацией о каждом коммите, такой как автор, дата, время и сообщение коммита.

14Ветвление (branching) - это создание отдельной линии разработки, которая может развиваться независимо от других веток. Оно позволяет работать над различными функциональными или исправительными задачами параллельно.

15Команда git checkout используется для переключения между ветками или восстановления файлов из предыдущих коммитов.

16Указатель HEAD - это ссылка на текущий коммит в Git. Он указывает на последний коммит в текущей ветке.

17Для создания новых веток в Git используйте команду git branch <branch-name>.

18Командаgit checkout <branch-name> используется для переключения на другую ветку.

19Команда git merge используется для объединения изменений из одной ветки в другую.

20Команда git log позволяет отобразить историю коммитов в репозитории Git с информацией о каждом коммите, включая автора, дату и сообщение коммита.

21 Чтобы выполнить слияние изменений из одной ветки в другую, нужно сначала переключиться на целевую ветку с помощью git checkout, а затем выполнить команду git merge <source-branch>, где <source-branch> - это имя ветки, из которой вы хотите внести изменения.

22 При слиянии могут возникать проблемы слияния (merge conflicts), когда Git не может автоматически объединить изменения из разных веток. Эти проблемы разрешаются путем ручного редактирования конфликтующих файлов и выбора правильной версии изменений.

qwerty@DESKTOP-0FNJ43D MINGW64 ~/Desktop/vrpo (master)

$ git branch

NEWB

\* master

qwerty@DESKTOP-0FNJ43D MINGW64 ~/Desktop/vrpo (master)

$ git status

On branch master

Untracked files:

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

1.log

test2.html

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

qwerty@DESKTOP-0FNJ43D MINGW64 ~/Desktop/vrpo (master)

$ git add test2.html

warning: in the working copy of 'test2.html', LF will be replaced by CRLF the next time Git touches it

qwerty@DESKTOP-0FNJ43D MINGW64 ~/Desktop/vrpo (master)

$ echo 1111 > test2.html

qwerty@DESKTOP-0FNJ43D MINGW64 ~/Desktop/vrpo (master)

$ git add test2.html

warning: in the working copy of 'test2.html', LF will be replaced by CRLF the next time Git touches it

qwerty@DESKTOP-0FNJ43D MINGW64 ~/Desktop/vrpo (master)

$ git commit -m "gg"

[master c6ef765] gg

1 file changed, 1 insertion(+)

create mode 100644 test2.html

qwerty@DESKTOP-0FNJ43D MINGW64 ~/Desktop/vrpo (master)

$ git branch NEWB

fatal: a branch named 'NEWB' already exists

qwerty@DESKTOP-0FNJ43D MINGW64 ~/Desktop/vrpo (master)

$ git checkout NEWB

Switched to branch 'NEWB'

qwerty@DESKTOP-0FNJ43D MINGW64 ~/Desktop/vrpo (NEWB)

$ git status

On branch NEWB

nothing to commit, working tree clean

qwerty@DESKTOP-0FNJ43D MINGW64 ~/Desktop/vrpo (NEWB)

$ echo yuio > test2.html

qwerty@DESKTOP-0FNJ43D MINGW64 ~/Desktop/vrpo (NEWB)

$ git status

On branch NEWB

Untracked files:

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

test2.html

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

qwerty@DESKTOP-0FNJ43D MINGW64 ~/Desktop/vrpo (NEWB)

$ git add test2.html

warning: in the working copy of 'test2.html', LF will be replaced by CRLF the next time Git touches it

qwerty@DESKTOP-0FNJ43D MINGW64 ~/Desktop/vrpo (NEWB)

$ git commit -m "welldone"

[NEWB eb36d60] welldone

1 file changed, 1 insertion(+)

create mode 100644 test2.html

qwerty@DESKTOP-0FNJ43D MINGW64 ~/Desktop/vrpo (NEWB)

$ git checkout master

Switched to branch 'master'

qwerty@DESKTOP-0FNJ43D MINGW64 ~/Desktop/vrpo (master)

$ git merge NEWB

Auto-merging test2.html

CONFLICT (add/add): Merge conflict in test2.html

Automatic merge failed; fix conflicts and then commit the result.

qwerty@DESKTOP-0FNJ43D MINGW64 ~/Desktop/vrpo (master|MERGING)

$

1. Что такое система контроля версий, для чего ее используют?
2. Какой принцип хранения файлов использует Git?
3. В чем отличие Git от других систем контроля версий?
4. В каких трех основных состояниях файлы могут находиться в Git ?
5. Что такое индексация файла? Какой командой она выполняется?
6. Что такое фиксация файла? Какой командой она выполняется?
7. Продемонстрируйте команду проверки выбранных настроек.
8. Как инициализировать репозиторий в существующей папке?
9. Как указать файлы, за которыми должна следить система?
10. Как выполнить фиксацию изменений?
11. Какой командой определить состояния файлов?
12. Для чего создается файл .gitignore? Поясните его структуру.
13. Как используется команда git log?
14. Что такое ветвление?
15. Как используется команда git checkout?
16. Что такое указатель HEAD?
17. Продемонстрируйте создание новых веток в Git?
18. Какая команда осуществляет переключение веток?
19. Для чего используется команда merge?
20. Какая команда позволяет отобразить историю репозитория?
21. Как выполнить слияние изменений из одной ветки в другую?
22. Какие проблемы могут быть при слиянии и как они разрешаются?

1. Система контроля версий (СКВ) - это инструмент, который используется для управления изменениями в файловой системе. СКВ позволяет отслеживать изменения в файлах и папках, восстанавливать предыдущие версии файлов, сравнивать изменения, объединять изменения от нескольких авторов и многое другое. Она используется для упрощения и улучшения совместной работы над проектами, отслеживания истории изменений, обеспечения безопасности и восстановления данных.

2. Git использует принцип хранения файлов в виде снимков (snapshots). При каждом коммите Git сохраняет полный набор файлов, которые представляют собой снимок текущего состояния проекта. При этом Git эффективно хранит только разницу (дельту) между последующими коммитами, что позволяет сэкономить место на диске.

3. Отличительные особенности Git от других систем контроля версий включают:

- Распределенная архитектура: Каждый локальный репозиторий Git содержит полную историю проекта, что позволяет работать независимо от сети и обеспечивает высокую гибкость и скорость работы.

- Быстрота: Git обладает высокой производительностью и эффективно обрабатывает как небольшие, так и очень большие проекты.

- Ветвление и слияние: Git предоставляет мощные инструменты для создания и управления ветками, что упрощает работу в команде и экспериментирование с разными версиями кода.

- Целостность данных: Все данные в Git хешируются, что обеспечивает целостность и проверку целостности данных.

4. Файлы в Git могут находиться в трех основных состояниях:

- Измененные (Modified): Файл был изменен, но еще не добавлен в индекс.

- Индексированные (Staged): Изменения файла были добавлены в индекс (подготовлены к коммиту).

- Зафиксированные (Committed): Изменения файла были зафиксированы в репозитории Git.

5. Индексация файла в Git - это процесс добавления изменений файла в индекс (stage), чтобы подготовить их к будущему коммиту. Индексация выполняется с помощью команды `git add <file>`. Например, `git add myfile.txt` добавляет файл `myfile.txt` в индекс.

6. Фиксация файла (или коммит) в Git - это сохранение изменений в репозитории Git. Фиксация создает новый коммит, который содержит снимок текущего состояния проекта. Фиксация выполняется с помощью команды `git commit -m "Commit message"`. Например, `git commit -m "Add new feature"` создает коммит с сообщением "Add new feature".

7. Команда `git config --list` отображает выбранные настройки Git, включая глобальные и локальные настройки пользователя, а также настройки репозитория.

8. Для инициализации репозитория в существующей папке нужно перейти в папку через командную строку и выполнить команду `git init`. Это создаст новый репозиторийв текущей папке.

9. Чтобы указать файлы, за которыми должна следить система, можно использовать команду `git add <file>` для каждого файла, который вы хотите добавить в репозиторий. Например, `git add myfile.txt` добавит файл `myfile.txt` для отслеживания.

10. Для выполнения фиксации изменений нужно использовать команду `git commit -m "Commit message"`. Здесь "Commit message" представляет собой описание изменений, внесенных в коммите.

11. Команда `git status` позволяет определить состояния файлов в репозитории Git. Она показывает, какие файлы были изменены, добавлены в индекс или зафиксированы.

12. Файл `.gitignore` создается для указания Git'у, какие файлы и папки следует игнорировать при отслеживании изменений. Он содержит шаблоны, которые определяют, какие файлы и папки должны быть проигнорированы. Например, можно добавить `\*.log` в `.gitignore`, чтобы игнорировать все файлы с расширением `.log`.

13. Команда `git log` используется для просмотра истории коммитов в репозитории Git. Она отображает список коммитов в обратном хронологическом порядке, показывая информацию о каждом коммите, такую как хеш коммита, автор, дата и сообщение коммита.

14. Ветвление в Git - это создание новой линии разработки, которая может отличаться от основной линии (ветки). Ветка позволяет работать над определенной функциональностью или исправлением ошибки, не затрагивая основную линию. Это позволяет изолированно разрабатывать и тестировать новый код, а затем объединить его с основной линией при слиянии.

15. Команда `git checkout` используется для переключения между ветками или восстановления файлов из предыдущих коммитов. Например, `git checkout branchname` переключается на ветку с именем `branchname`, а `git checkout -- myfile.txt` восстанавливает файл `myfile.txt` из последнего коммита.

16. Указатель HEAD - это указатель на текущий коммит в ветке. Он используется для обозначения последнего коммита и текущего положения в истории проекта.

17. Для создания новой ветки в Git используется команда `git branch <branchname>`. Например, `git branch feature` создает новую ветку с именем `feature`.

18. Команда `git checkout <branchname>` осуществляет переключение на указанную ветку. Например, `git checkout feature` переключается на ветку `feature`.

19. Команда `git merge` используется для объединения изменений из одной ветки в другую. Она берет изменения из указанной ветки и интегрирует их в текущую ветку. Например, `git merge feature` объединяет изменения из ветки `feature` в текущую ветку.

20. Команда `git log` позволяет отобразить историю коммитов в репозитории Git. Она показывает список коммитов с информацией о каждом коммите, такой как хеш коммита, автор, дата и сообщение коммита.

21. Чтобы выполнить слияние изменений из одной ветки в другую, нужно сначала переключиться на целевую ветку (напр