Список команд **POWERSHELL**

Узнать список файлов текущего каталога **DIR**

Узнать точный путь до текущей папки можно при помощи команды **pwd**

Для перемещения между каталогами используется команда **cd**

Создать файл можно при помощи команды **touch** (touch hello.txt)

**TYPE** для вывода содержимого файла ??

Для создания каталога можно воспользоваться командой **mkdir** (mkdir test)

Можно перенаправить текст в файл, для этого можно воспользоваться символом больше >, после которого указывается имя файла **echo 'Hello, world!' > hello.txt**

Содержимое файла мы можем прочитать при помощи команды **cat** (cat hello.txt)

Чтобы дозаписать какую-то информацию в конец файла, не перезатирая его содержимое, можно воспользоваться последовательностью два знака больше **>>**: (echo 'Hello, world!' >> hello.txt)

Удалить файл hello.txt можно при помощи команды **unlink (unlink**

Список команд в командной строке

ASSOC Вывод либо изменение сопоставлений по расширениям имен файлов.

ATTRIB Отображение и изменение атрибутов файлов.

BREAK Включение и выключение режима обработки комбинации клавиш CTRL+C.

BCDEDIT Задает свойства в базе данных загрузки для управления начальной

загрузкой.

CACLS Отображение и редактирование списков управления доступом (ACL)

к файлам.

CALL Вызов одного пакетного файла из другого.

CD Вывод имени либо смена текущей папки.

CHCP Вывод либо установка активной кодовой страницы.

CHDIR Вывод имени либо смена текущей папки.

CHKDSK Проверка диска и вывод статистики.

CHKNTFS Отображение или изменение выполнения проверки диска во время

загрузки.

CLS Очистка экрана.

CMD Запуск еще одного интерпретатора командных строк Windows.

COLOR Установка цветов переднего плана и фона, используемых по умолчанию.

COMP Сравнение содержимого двух файлов или двух наборов файлов.

COMPACT Отображение и изменение сжатия файлов в разделах NTFS.

CONVERT Преобразует тома FAT в NTFS. Вы не можете

преобразовать текущий диск.

COPY Копирование одного или нескольких файлов в другое место.

DATE Вывод либо установка текущей даты.

DEL Удаление одного или нескольких файлов.

DIR Вывод списка файлов и подпапок из указанной папки.

DISKPART Отображает или настраивает свойства раздела диска.

DOSKEY Редактирует командные строки, повторно вызывает команды Windows и создает

макросы.

DRIVERQUERY Отображает текущее состояние и свойства драйвера устройства.

ECHO Отображает сообщения и переключает режим отображения команд на экране.

ENDLOCAL Завершает локализацию изменений среды для пакетного файла.

ERASE Удаляет один или несколько файлов.

EXIT Завершает работу программы CMD.EXE (интерпретатора командных строк).

FC Сравнивает два файла или два набора файлов и

отображает различия между ними.

FIND Ищет текстовую строку в одном или нескольких файлах.

FINDSTR Ищет строки в файлах.

FOR Запускает указанную команду для каждого из файлов в наборе.

FORMAT Форматирует диск для работы с Windows.

FSUTIL Отображает или настраивает свойства файловой системы.

FTYPE Отображает либо изменяет типы файлов, используемые при

сопоставлении по расширениям имен файлов.

GOTO Направляет интерпретатор команд Windows в отмеченную строку

пакетной программы.

GPRESULT Отображает информацию о групповой политике для компьютера или пользователя.

GRAFTABL Позволяет Windows отображать расширенный набор символов в

графическом режиме.

HELP Выводит справочную информацию о командах Windows.

ICACLS Отображает, изменяет, архивирует или восстанавливает

списки ACL для файлов и каталогов.

IF Выполняет условную обработку в пакетных программах.

LABEL Создает, изменяет или удаляет метки тома для дисков.

MD Создает каталог.

MKDIR Создает каталог.

MKLINK Создает символьные ссылки и жесткие связи

MODE Настраивает системные устройства.

MORE Последовательно отображает данные по частям размером в один экран.

MOVE Перемещает один или несколько файлов из одного каталога

в другой.

OPENFILES Отображает файлы, открытые для файлового ресурса удаленными пользователями.

PATH Отображает или устанавливает путь поиска исполняемых файлов.

PAUSE Приостанавливает выполнение пакетного файла и выводит сообщение.

POPD Восстанавливает предыдущее значение текущего каталога,

сохраненное с помощью команды PUSHD.

PRINT Выводит на печать содержимое текстового файла.

PROMPT Изменяет командную строку Windows.

PUSHD Сохраняет текущий каталог, затем изменяет его.

RD Удаляет каталог.

RECOVER Восстанавливает данные, которые можно прочитать, с плохого или поврежденного диска.

REM Записывает комментарии в пакетные файлы или файл CONFIG.SYS.

REN Переименовывает файлы.

RENAME Переименовывает файлы.

REPLACE Заменяет файлы.

RMDIR Удаляет каталог.

ROBOCOPY Улучшенная служебная программа копирования файлов и деревьев папок

SET Показывает, устанавливает или удаляет переменные среды Windows.

SETLOCAL Начинает локализацию изменений среды в пакетном файле.

SC Отображает или настраивает службы (фоновые процессы).

SCHTASKS Выполняет команды и запускает программы на компьютере по расписанию.

SHIFT Изменяет положение заменяемых параметров в пакетных файлах.

SHUTDOWN Позволяет локально или удаленно завершить работу компьютера.

SORT Сортирует ввод.

START Выполняет указанную программу или команду в отдельном окне.

SUBST Связывает путь с именем диска.

SYSTEMINFO Отображает сведения о свойствах и конфигурации определенного компьютера.

TASKLIST Отображает все выполняемые задачи, включая службы.

TASKKILL Прекращение или остановка процесса либо приложения.

TIME Отображает или устанавливает системное время.

TITLE Назначает заголовок окна для сеанса CMD.EXE.

TREE Графически отображает структуру каталогов диска или

пути.

TYPE Отображает содержимое текстовых файлов.

VER Отображает сведения о версии Windows.

VERIFY Устанавливает режим проверки в Windows правильности записи

файлов на диск.

VOL Отображает метку и серийный номер тома для диска.

XCOPY Копирует файлы и деревья папок.

WMIC Отображает сведения об инструментарии WMI в интерактивной командной оболочке.

Работа с GIT

**git –version** – версия git, установленная на компьютере

Перед началом работы необходимо внести все данные

git config --global user.name "Igor Simdyanov"

git config --global user.email igorsimdyanov@gmail.com

Убедиться в том, что настройки успешно установлены, можно, запросив их список при помощи команды **git config --list.**

Определить местоположение конфигурационного файла можно, если добавить к команде параметр **--show-origin**

git config --list --show-origin

Создание git-проекта

Git следит за состоянием не всей файловой системы, а выбранной папки на компьютере. Такую папку необходимо пометить как Git-проект.

На этом этапе файлы проекта пока не находятся под версионированием системы контроля версий. Давайте добавим, какой-либо файл, например создадим новый файл hello.txt. Сделать этого можно либо в текстовом редакторе, либо, если мы находимся в UNIX-подобной операционной системе, при помощи команды touch:

touch hello.txt

Убедиться, что файл успешно создан, можно при помощи команды ls:

ls -la

Чтобы добавить файл под контроль git, необходимо проиндексировать его при помощи команды git add:

**git add .**

При этом точка означает все новые файлы в папке, вместо этого можно указать избранные файлы или каталоги:

git add hello.txt

Такая команда добавит только данный файл

После того, как мы сообщили Git, какие файлы проекта мы хотим отслеживать, их можно зафиксировать. Для этого создается снимок состояния, который называется коммитом. Сделать это можно при помощи команды git commit:

$ git commit -m 'Первый коммит'

В параметре -m мы задаем комментарий к коммиту

Взаимодействие с редактором vim может быть довольно утомительно, так как для комфортной работы потребуется изучить его режимы и многочисленные клавиатурные сокращения. Если работа с редактором vim вас пугает, можете либо каждый раз добавлять параметр -m и задавать комментарий в команде git commit, либо настроить другой редактор по умолчанию. Например, в качестве такого редактора можно задать nano, который гораздо проще, чем vim.

Проверяем статус git-проекта при помощи команды git status

Есть отслеживаемые и неотслеживаемые файлы.

Если внести изменения в файл отслеживаемый, сохранить данные изменения, то при проверке статуса этот документ будет также выделен красным (также, как и неотслеживаемые)

Вывод команды git status довольно объемный, можно добиться более компактного вывода, если добавить ей параметр -s (git status -s)

?? новые неотслеживаемые файлы

M отредактированные файлы

Гит анализирует только отслеживаемые файлы

Если после того, как внесли изменения и проиндексировали файлы, вызвать команду git diff, то ничего не произойдет

НО!!! если мы добавим в команду параметр --cached, мы можем увидеть изменения, которые сейчас находятся в индексе

**git diff --cached**

Если файл неотслеживаемый, то Про неотслеживаемый файл git ничего не знает, поэтому, чтобы от него избавиться, достаточно его просто удалить (в ручную или командой **unlink** to\_delete.txt)

С файлом hello.txt так уже не получится, он отслеживаемый. Кроме того, мы не хотим удалять файл, мы хотим получить последнее сохраненное состояние в отчете, и git нам уже подсказывает что делать дальше с файлом: ● либо индексировать при помощи команды git add, ● либо откатывать изменения при помощи команды git checkout --.

git checkout -- hello.txt

для того, чтобы удалить индексируемый файл, нужно git reset HEAD

git reset HEAD hello.txt

данная команда отправляет файл в неиндексируемое состояние

для того, чтобы откатиться к последнему изменению git checkout – hello.txt

Посмотреть историю изменений можно при помощи команды git log

(от более свежих до более ранних)

Если коммитов много, они не убираются на экране, двигаться дальше можно, нажимая на пробел. Выйти из режима просмотра можно, нажав q

По умолчанию для каждого коммита выводится хэш, автор, время добавления и комментарий к коммиту. Ограничить количество выводимых коммитов можно при помощи параметра -2.

git log -2

По умолчанию в отчете команды выводится только метаинформация. Чтобы вывести детальные изменения для каждого из коммитов, команде следует передать параметр -p

git log -2 -p

Если нужна статистика для каждого коммита, можно воспользоваться параметром --stat:

git log -2 –stat

Под каждым из коммитов выводится список измененных файлов, а также количество строк, которое было добавлено и удалено в каждом из файлов. Каждая добавленная строка обозначается зеленым плюсом, а каждая удаленная — красным минусом. Еще больше сократить вывод команды git log можно при помощи параметра --pretty, который изменяет формат вывода

git log --pretty=oneline

Например, таким образом можно добиться, чтобы коммиты выводились в одну строку. Можно последовательно почитать комментарии к коммитам в хронологическом порядке. Если вместо oneline использовать format, можно сформировать свой собственный формат логов:

git log --pretty=format:"%h - %an, %ar : %s"

Как мы видели в выводе команды git log, каждый коммит снабжается хэш-суммой. Механизм, которым пользуется Git при вычислении хеш-сумм, называется SHA-1 хеш. Это строка длинной в 40 символов в шестнадцатеричном формате, каждая позиция может принимать либо числовое значение от 0 до 9, либо английский символ от a до f. Т. е., каждая из позиций кодирует 16 значений.

Для поиска коммитов в определенном временном интервале можно использовать параметры --since и --until: ● --until позволяет задать дату, с которой мы начинаем поиск, ● --since — задать, до какого момента в прошлом мы ищем коммиты. Команда ниже позволяет искать коммиты, начиная двумя неделями назад и до текущего момента:

git log --since=2.week

Следующая команда ищет коммиты с начала проекта до даты недельной давности:

git log --until=1.week

Допускается использовать и точные даты.

$ git log --since=2019-03-01 --until=2019-03-25

При помощи параметра -S можно найти коммит, в котором была добавлена или удалена заданная строка:

$ git log -SGit -p

Команда выше ищет коммиты, в которых упоминается слово Git.

Иногда необходимо найти коммиты, которые принадлежат конкретному разработчику. Сделать это можно при помощи параметра --author:

$ git log --author="Igor Simdyanov"

Иногда стоит обратная задача, нужно, наоборот, выяснить, кто автор этой строки в файле и в каком коммите она в последний раз менялась Для этого предназначена команда git blame:

$ git blame readme.txt

В качестве параметра команда принимает имя файла. В результате выводится содержимое файла, в котором каждая строка помечена хэшем коммита, автором и датой последнего изменения.

Чтобы Git игнорировал нежелательные файлы, предусмотрен специальный конфигурационный файл .gitignore, который помещается в корень проекта. Внутри файла прописывается шаблон: файлы, которые удовлетворяют им, игнорируются Git. \*.log tmp/\*

Например, так мы будем игнорировать все файлы, которые завершаются расширением log, а также файлы в папке tmp.

Ресурс http://gitignore.io/ позволяет автоматически формировать содержимое .gitignore-файла. Достаточно указать ключевые слова, например Sublime, Yii2, vim, и на их основе будет предложена заготовка для .gitignore, которая исключает временные и вспомогательные файлы.

Git ориентируется на файлы, папки попадают под отслеживание, если только в них находятся файлы, пустые папки — игнорируются. Давайте добавим пустую папку src в проект и запросим статус проекта при помощи команды git status:

$ git status В результате папка не попадает в индекс. Это не всегда удобно, иногда папки нужны для функционирования проекта. Например, приложение может ожидать наличие папки log, но так как мы запретили log-файлы, такая папка рассматривается как пустая и не попадает в индекс git-проекта. Чтобы избежать этого, в папку часто добавляют файл .keep. Это просто обычный пустой файл, он не является частью экосистемы git и вместо него можно использовать любой другой файл. Файл начинается с точки, поэтому в UNIX-подобных операционных системах он рассматривается как скрытый. В Windows это не так, однако это название получило широкое распространение в тех случаях, когда важно сохранить пустую папку в git-проекте.

Система контроля версий мало бы отличалась от системы резервного копирования, если бы позволяла только делать снимки состояния проекта. Особенностью таких систем является возможность создания нескольких веток, которые, в свою очередь, являются основой для командной работы над проектом

Сразу после создания проекта появляется основная ветка master:

$ git branch \* master

В этом можно убедиться при помощи команды git branch. Данная команда выводит список доступных веток, помечая текущую звездочкой: пока у нас только одна ветка maste