Инструментальные средства системного администрирования

Устройство FreeBSD. Утилиты, сетевые утилиты FreeBSD. Установка из портов, из пакетов, сборка пакета.

[Введение](#_c713xnuw3xem)

[Компоненты ОС FreeBSD](#_819ud2jqeqjp)

[Интерфейс системных вызовов (System Call Interface)](#_g2vudqp3fb9)

[Виртуальная файловая система (VFS)](#_7hna73vzwxy9)

[UFS (Unix File System)](#_ybi7pz70whyp)

[ZFS (Zettabyte File System)](#_r7q5nn9nnf02)

[GEOM](#_tytp9dnj4i04)

[Block Device Interface (Интерфейс блочных устройств)](#_1nhcwhfodj3c)

[Стек TCP/IP](#_uzidu4wibqh4)

[Сокеты (Sockets)](#_png28ahxr10v)

[TCP/UDP](#_9rceq6dkxr3o)

[IP](#_gvz3e6fb91a7)

[Ethernet](#_qubdgqxex1ef)

[Драйверы устройств (Device Drivers)](#_2k23a4aolel5)

[Планировщик (scheduler)](#_dmpma12evwv2)

[Виртуальная память (Virtual Memory)](#_nnwj9qar2spz)

[Утилиты](#_gxf3h2pmnu2b)

[ipmitool](#_lh837wgmpm2)

[pstat -s](#_izhfmyp83brg)

[iostat](#_d4pvhjv0diiy)

[pmcstat](#_69tu9j9hrdki)

[vmstat](#_g9h23tybjmq8)

[gstat](#_p9l08jyqks77)

[fstat](#_j4xh6qnvxa87)

[systat](#_lq7zm38zei6g)

[vmstat](#_q3dmrasuin46)

[ps](#_b1ihw43twd3b)

[top](#_ooy2mzdhucbd)

[ktrace](#_xar8o1lb3oia)

[kdump](#_4cu20xf6dhii)

[truss](#_c2jie8jx8r5x)

[lockstat](#_j0d6juuulxbs)

[htop](#_9tp265rk3rzc)

[DTrace](#_696nutrwrdbq)

[Сетевые утилиты](#_xpbrazvnedji)

[netstat](#_xnn1doamy9qg)

[systat -ifstat](#_h95u327xx1h7)

[sockstat](#_585kqvk7nzp6)

[tcpdump](#_vkd5y7ltk462)

[iperf](#_srwrzhy2hbnf)

[mtr](#_xihxrswrgd2r)

[Установка ПО](#_n9dmw74n6zo6)

[Порты](#_jo2m6tyg7p8x)

[pkg](#_35jeh1dh0bv8)

[Соберем пакет из порта на примере nginx](#_tmaa9fir0vq0)

[portdowngrade](#_t51tjyrxnlsi)

[Домашнее задание](#_3tfrjxxltv85)

[Дополнительные материалы](#_y937sk8fclye)

[Используемая литература](#_uvp6qax5r1ok)

# 

# Введение

Многие инструменты схожи с теми, которые знакомы вам по Linux. Но могут быть и отличия, в частности, могут отличаться ключи запуска.

Вспомним, что современные процессоры обычно поддерживают 4 уровня выполнения, из которых операционные системы используют, как правило, два: привилегированный режим, в котором выполняется ядро операционной системы, и непривилегированный (пользовательское пространство). Операционная система предоставляет механизм системных вызовов, который позволяет прикладным программам получать доступ к услугам ядра операционной системы.

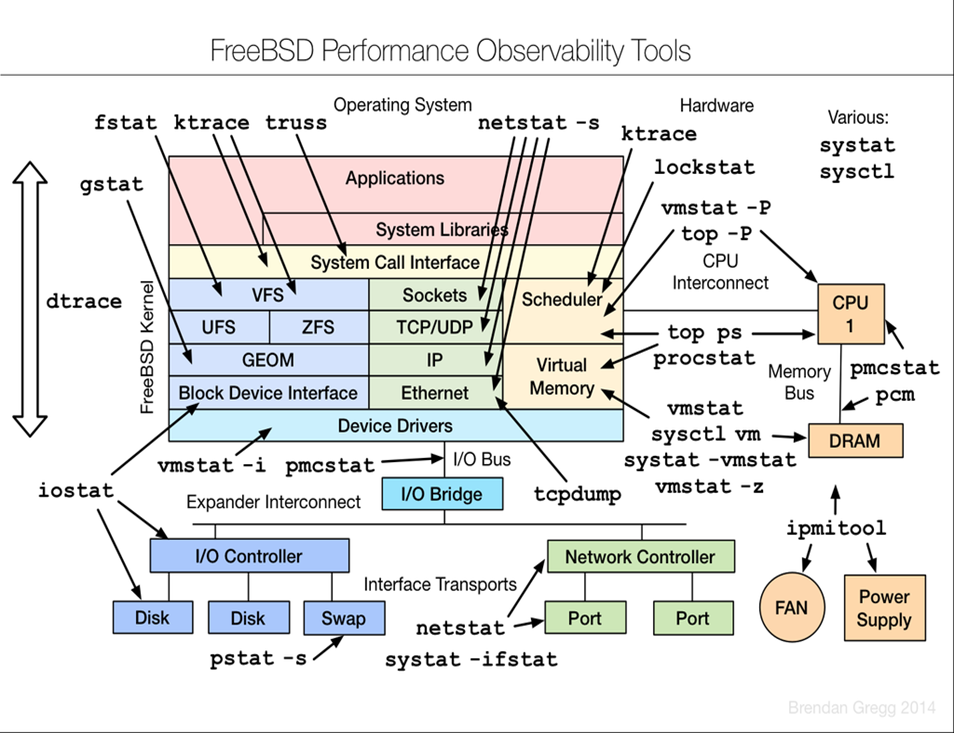
Так же, как и у Linux, ядро FreeBSD – монолитное модульное, исполняемое в привилегированном режиме, большинство программ выполняются в пространстве пользователя (непривилегированном режиме). Как и в большинстве современных операционных систем, приложения пользовательского пространства и ядра взаимодействуют благодаря **интерфейсу системных вызовов (System Call Interface).** **Приложения (Applications)** могут как напрямую использовать системные вызовы, так и подключать **системные библиотеки (System libraries),** которые также работают в пользовательском пространстве, но упрощают разработку и использование.

Отличие пространства пользователя и пространства ядра следует уже из названия соответствующих режимов работы процессора: непривилегированный и привилегированный. Ядро может обращаться к оборудованию, управлять памятью, иметь полный доступ к системе. Код, работающий в непривилегированном режиме, не может напрямую взаимодействовать с оборудованием, имеет доступ не ко всей памяти, а только к выделенной ему, и для взаимодействия с оборудованием и другими низкоуровневыми ресурсами обращается к ядру через механизм системных вызовов.

Стоит отметить, что переключение контекста при переходе из одного режима в другой (а оно происходит при каждом системном вызове и при возврате управления пользовательской программе) требует определенных затрат. Поэтому ряд компонентов, для которых критически важно быстродействие, реализуется в пространстве ядра (например стек TCP/IP). С другой стороны, ряд компонентов, требующих работы с оборудованием (файловая система, управление задачами и виртуальной памятью, драйверы устройств и т. д.) в принципе может работать только в привилегированном режиме. Технически возможно вынести часть кода драйвера в пространство пользователя. Так, до недавнего времени для системы GNU/Linux по лицензионным соображениям отсутствовал драйвер ZFS в виде модуля ядра, и подключить его можно было только через модуль FUSE, при этом сама реализация ZFS оставалась в пользовательском пространстве. Это решение обеспечивает не лучшее быстродействие. В Ubuntu 16 можно использовать ZFS благодаря драйверу, реализованному как модуль ядра.

# Компоненты ОС FreeBSD

Рассмотрим некоторые компоненты FreeBSD. Каждая из упомянутых далее утилит, как правило, работает с тем или иным компонентом операционной системы.



## Интерфейс системных вызовов (System Call Interface)

Долгое время для механизма системных вызовов применялся шлюз системных вызовов, реализованный как одно из программных прерываний (interrupt). В случае программного прерывания программа сама приостанавливает работу вызовом машинной инструкции int, вызывая обработчик программного прерывания. Перед этим в регистрах и стеке могут передаваться данные и указатели, аналогичным образом возвращается результат. Это похоже на вызов функций в языках программирования. Так же, как и Linux, FreeBSD использует номер прерывания 0x80. Таким образом, непосредственно передача управления ядру выполняется машинной командой int 0x80. При этом в 32-битной архитектуре (начиная с Pentium II) появилась новая машинная инструкция специально для осуществления системного вызова – sysenter, а в 64-битной архитектуре – syscall. Любое обращение к файлам, устройствам, сети производится через системный вызов. Что интересно, в 32-битном коде может применяться только sysenter, в – 64-битном – syscall, а int 0x80 – и там и там (хотя и работает чуть медленнее).

## Виртуальная файловая система (VFS)

Знакома многим по Linux и по расхожему утверждению, что в UNIX «все есть файл». VFS – дополнительный уровень абстракции, позволяющий работать с разными файловыми системами, физическими носителями, сетевыми носителями и т. д. благодаря тому, что сама работа с ними осуществляется на более низком уровне, в пространстве ядра, соответствующими модулями. Таким образом, работа через единую иерархию VFS возможна и с накопителями UFS, ZFS, и с сетевыми устройствами, с именованными каналами и т. д.

## UFS (Unix File System)

Файловая система, применяемая во FreeBSD, похожа на ext2 (и не случайно: ext2 создана по образцу UFS). Возможности (права, ссылки и т. д.) аналогичны ext2. Есть возможность журналирования (см. далее).

## ZFS (Zettabyte File System)

Изначально появилась в Solaris. В Linux при использовании более одного устройства для организации раздела требуется менеджер томов (например LVM – Logical Volume Manager). В ZFS для этого используются пулы устройств. Также поддерживается создание снимков (снэпшотов).

## GEOM

В отличие от Linux во FreeBSD файловые системы (UFS, ZFS) работают поверх GEOM – модульного фреймворка для хранилищ (storage). GEOM позволяет подключать модули для решения тех или иных задач: организации программного RAID-массива (geom\_mirror, geom\_raid и т. д.), шифрования (geom\_eli и т. д.), кеширования (geom\_cache), журналирования (geom\_journal), работы поверх LVM2 (geom\_linux\_lvm) и т. д.

## Block Device Interface (Интерфейс блочных устройств)

Так же, как и в Linux, система работает с блочными устройствами с произвольным доступом, представленными в виде файлов в VFS, с которыми обменивается информацией в виде блоков данных. Интерфейс предоставляет унифицированный доступ к разнообразным устройствам, физическое и логическое взаимодействие с которыми уже обеспечивают драйверы соответствующих устройств **(Device Drivers).**

## Стек TCP/IP

В настоящее время реализован практически во всех операционных системах, но впервые он появился в ОС FreeBSD. Теоретически стек TCP/IP может быть реализован в пространстве пользователя (за исключением только доступа к самим сетевым устройствам), но практически везде реализуется на уровне ядра из соображений быстродействия.

Реализация стека TCP/IP включает:

* сокеты (Socket);
* TCP/UDP;
* IP;
* Ethernet.

## Сокеты (Sockets)

Механизм сокетов позволяет приложениям получать доступ к сетевым услугам, во многом похожий на доступ к файлам. Так же, как для доступа к файлу нужно получить его дескриптор, при работе с сетью необходимо получить дескриптор сокета (socket), где требуется указать семейство протоколов (UNIX-сокеты: unix, IPv4: inet, IPv6: inet6 и т. д.), тип сокета (stream – потоковый или dgram – дейтаграммный) и протокол. Далее необходимо выполнить привязку сетевого адреса и порта (bind) и начать прослушивание (listen) либо подключиться к удаленному хосту (connect). Чтение и запись (read и write) аналогичны чтению файла, хотя возможно использование и более характерных для сети механизмов (recv и send). После завершения работы сокет, как и файл, требуется закрыть (close). Для работы с TCP-соединениями используется IPv4 или IPv6 и stream-сокет. Для работы с UDP используется IPv4 или IPv6 и dgram-сокет. Если два процесса работают на одной машине, вместо TCP/IP через localhost можно использовать UNIX-сокеты. В этом случае вместо TCP или UDP будет использоваться файловый сокет. Он именуется как файл в **VFS,** но данные передаются через оперативную память от одного приложения к другому.

## **TCP/UDP**

TCP и UDP – два независимых протокола транспортного уровня, отвечающие за идентификацию приложений – отправителя и получателя (TCP-порты и UDP-порты, контроль целостности, для TCP – разбиение потока полученных от приложения данных на сегменты, сборка полученных сегментов получателем и механизм подтверждений). Напомним, пространства TCP-портов и UDP-портов независимы друг от друга, с каждым из них работа ведется отдельно.

## **IP**

Протокол сетевого уровня, инкапсулирует TCP-сегменты или UDP-дейтаграммы в IP-пакет, отвечающий за глобальную идентификацию хостов (IPv4- или IPv6-адресация), маршрутизацию (выбор маршрута по таблице маршрутизации) и пересылку сетевых пакетов (в случае, если машина работает в качестве маршрутизатора). Также на сетевом уровне имеется механизм уведомления о некоторых сетевых проблемах (маршрут не найден, TTL истек и т. д) – протоколы ICMP и ICMPv6.

## Ethernet

**Ethernet** – один из популярных протоколов канального уровня для работы по витой паре или оптоволокну, отвечающий за идентификацию интерфейсов в сети (MAC-адреса), инкапсуляцию IP-пакетов в Ethernet-кадры, расчет контрольной суммы и отправку данных через физическую среду передачи данных (например, витую пару). Последнее также возможно благодаря драйверам сетевых устройств **(Device Drivers).**

## Драйверы устройств **(Device Drivers)**

Драйверы устройств **(Device Drivers),** работая в пространстве ядра, обеспечивают взаимодействие с аппаратным обеспечением (дисковыми устройствами, сетевыми устройствами), используя **I/O Bus (шину ввода-вывода** – для передачи команд и данных в и из устройства) и механизм **аппаратных прерываний (IRQ – Interrupt Request:** при необходимости передачи данных устройство генерирует аппаратное прерывание, по которому процессор прерывает выполняемую в данный момент программу, сохраняет ее контекст, после чего передает управление обработчику прерывания соответствующего драйвера устройства). Приложения же взаимодействуют с драйверами неявным образом, например при работе с виртуальной файловой системой (VFS), при чтении-записи файлов, или использовании механизма сокетов (Sockets), либо обращаясь напрямую **к символьным** или **блочным устройствам** в файловой системе /dev.

Также в составе любой современной многозадачной операционной системы реализованы модули – планировщик (scheduler) и механизм управления виртуальной памятью (virtual memory).

## Планировщик (scheduler)

**Планировщик (scheduler)** – важный механизм реализации многозадачности. Он управляет процессами, их приоритетами, очередью заданий. Не стоит путать с cron, это не запуск процессов по расписанию, это именно управление исполняемым процессами кодом. Scheduler выделяет кванты времени процессора **(CPU – Central Processor Unit)** по очереди коду процессов, передавая им управление, с учетом приоритета и операций ввода-вывода.

## Виртуальная память (Virtual Memory)

**Виртуальная память (Virtual Memory)** – важный механизм работы, появившийся вместе с механизмом колец защиты. Теперь приложение не работает непосредственно с физической памятью, но адресует инструкции и данные в виртуальной памяти, которая отображается на физическую память **(DRAM – Dynamic Random Access Memory)**. При этом код и данные могут быть загружены в физическую память и принадлежать данному процессу или сразу нескольким (разделяемая, **shared память**), а также выгружены в раздел подкачки **(swap)**. Когда приложение попытается обратиться к области памяти, которая не загружена в этот момент, будет сформировано исключение, и управление перейдет компоненту ядра, который загрузит эту область памяти с диска (возможно, выгрузив на диск данные другого процесса, используемые в этот момент).

# Утилиты

## ipmitool

ipmitool позволяет снимать с помощью протокола **IPMI (Intelligent Platform Management Interface)** показатели, такие как управление вентилятором **(Fan Control),** питание **(Power Supply).** Данные могут быть использованы, например, в системе мониторинга Zabbix. Стоит отметить, что эта программа может пригодиться только на физических машинах (особенно сейчас, когда распространены системы виртуализации).

## pstat -s

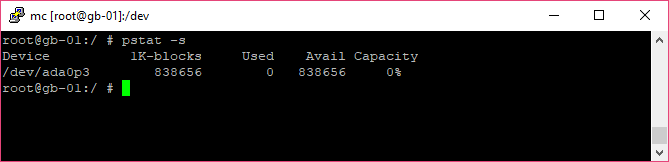
pstat – утилита, которая позволяет получать статистику по структурам системы, таким как открытые файлы, используемые терминалы и т. д.

Чтобы посмотреть использование свопа (раздела подкачки), можно использовать:

|  |
| --- |
| pstat -s |

или, что то же самое:

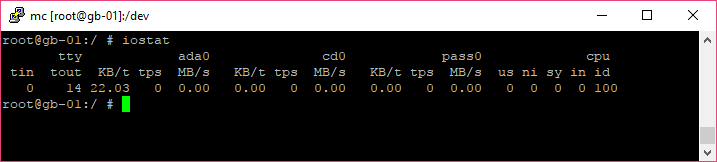
|  |
| --- |
| swapinfo |



## iostat

Утилита, позволяющая отслеживать статистику ввода/вывода, в особенности по жестким дискам.

По умолчанию выдача программы скудна:

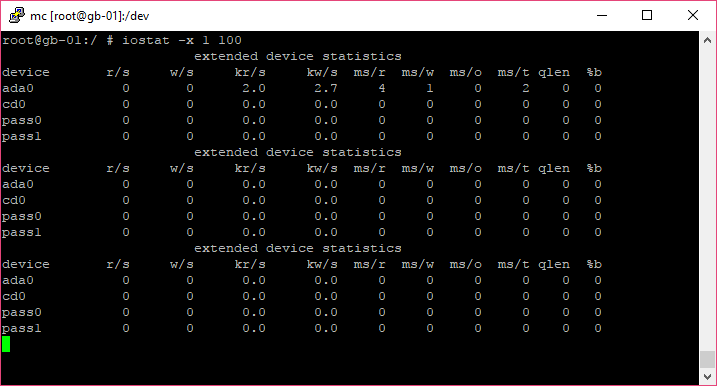


Можно запустить в режиме top с ключом -x, указав интервал снятия данных и количество повторов.

Каждую

секунду – 100 снятий:

|  |
| --- |
| iostat -x 1 100 |



Выдача несколько отличается от аналогичной в Linux.

Последняя строка – процент занятости.

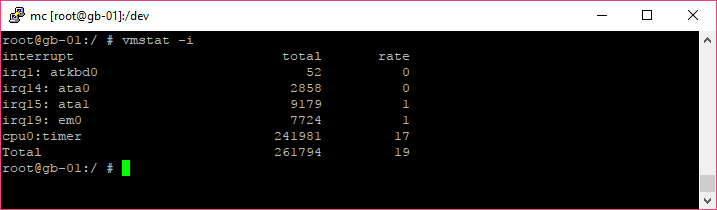
## pmcstat

Утилита позволяет измерить производительность системы с учетом производительности аппаратной составляющей (hardware).

## vmstat

vmstat – утилита, показывающая статистику ядра по процессам, виртуальной памяти, диску, прерываниями и т. д.

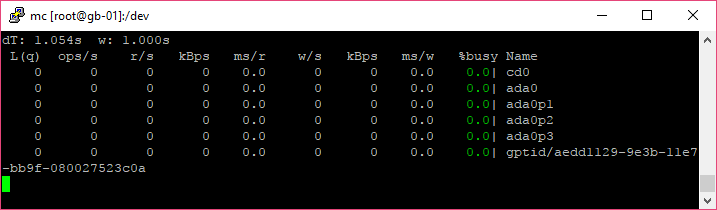
Например, ключ -i позволяет вывести информацию о прерываниях по устройствам.



## gstat

gstat – статистика по GEOM.

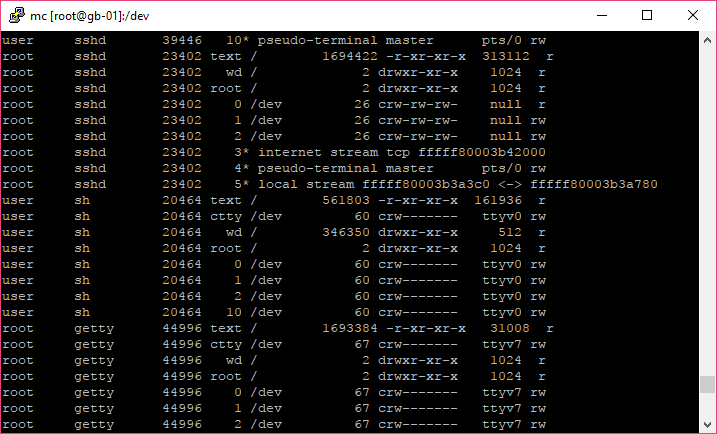
|  |
| --- |
| gstat |



## fstat

fstat – статистика по открытым файлам.

|  |
| --- |
| fstat |



Обратите внимание, что TCP/IP-соединения также отображаются среди открытых файлов.

## systat

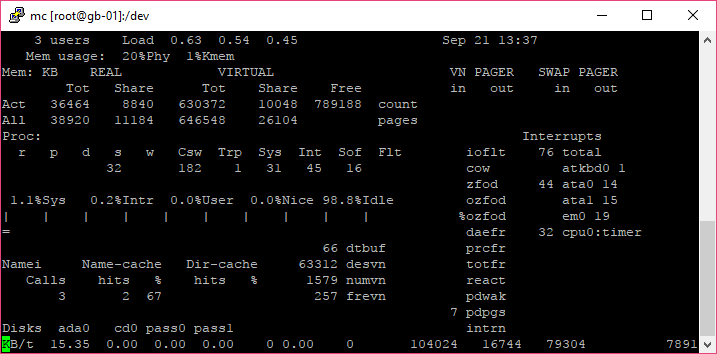
systat – утилита, позволяющая узнать информацию о многих компонентах системы.

Перечень ключей:

|  |
| --- |
| man systat |

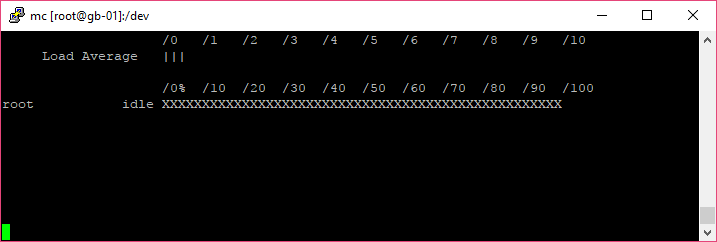
Чтобы вывести статистику по виртуальной памяти, используем с ключом -vmstat:

|  |
| --- |
| systat -vmstat |



Интересен ключ iostat: запущенная с этим ключом systat становится аналогом iotop из linux.

|  |
| --- |
| systat -iostat |

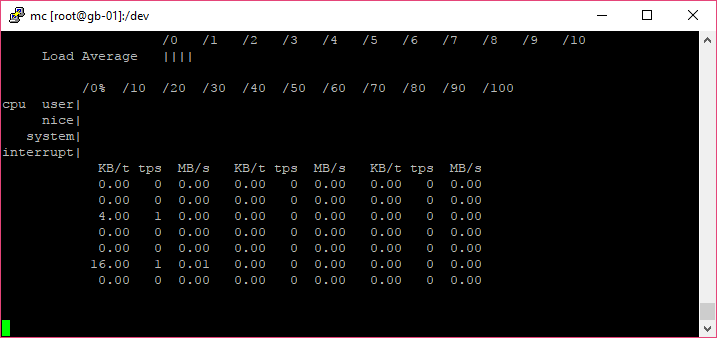


Статистика по диску будет выводиться в реальном времени.

Более подробно:

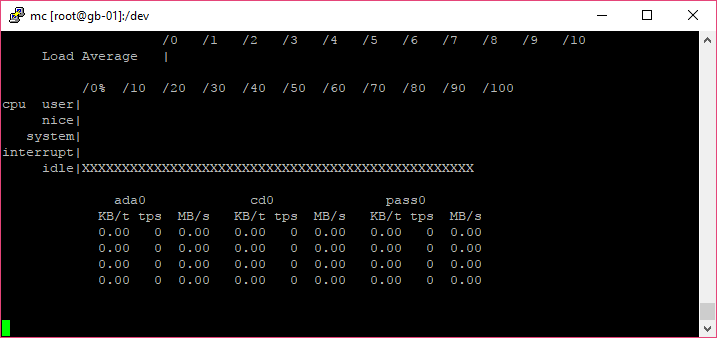
|  |
| --- |
| systat -iostat -numbers -- 2.1 |

Будет отображаться статистика ввода-вывода в цифровом формате, и информация будет обновляться каждые 2,1 секунды.



С отображением для доступных устройств:

|  |
| --- |
| systat -iostat -numbers -- 2.1 drives |

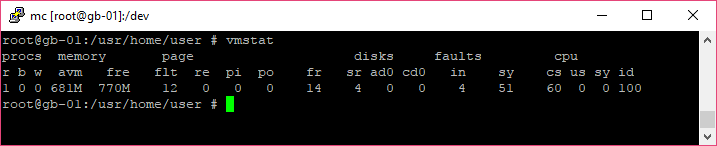


Более подробно:

|  |
| --- |
| man systat |

## vmstat

Утилита vmstat выводит статистику по использованию виртуальной памяти.



## ps

Отображает информацию о выполняемых процессах.

Примеры:

|  |
| --- |
| ps aux |

-a выводит информацию и для процессов других пользователей.

-u отображает параметры: user, pid, %cpu, %mem, vsz, rss, tt, state, start, time и command.

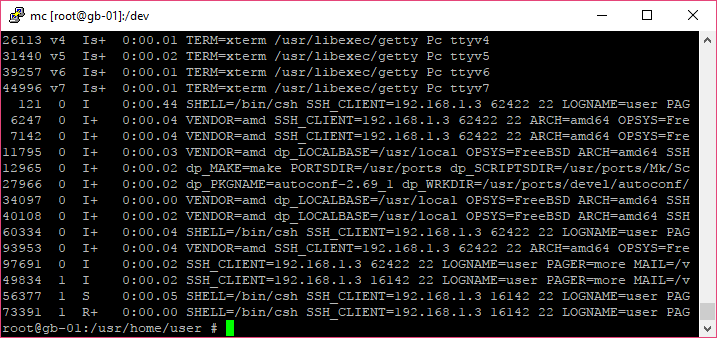
-x отображает процессы, отсоединенные от терминала.

## 

|  |
| --- |
| ps ef |

-e отображает среду;

-f показывает информацию о командной строке и окружении процесса. Будет выполняться, только если UID=0 (т. е. для root).



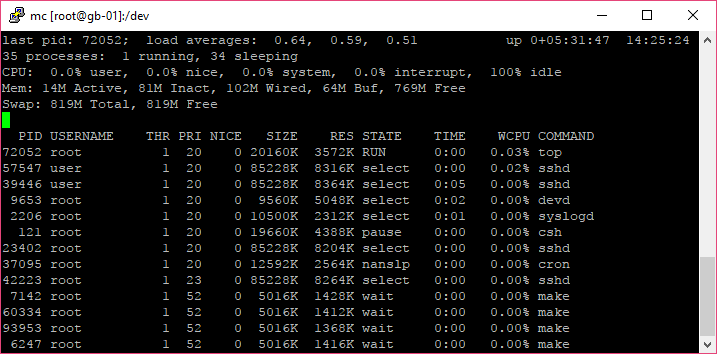
Обозначения состояния процессов (state):

* D – процесс в ожидании диска (или аналогичном непрерываемом ожидании);
* I – idle (процесс спит более 20 секунд);
* L – процесс, ожидающий получения блокировки;
* R – запущенный процесс;
* S – процесс спит менее 20 секунд;
* T – остановленный процесс;
* W – прерывание по ожиданию;
* Z – процесс-зомби.

Также дополнительный символ может означать:

* + – процесс выполняется foreground для терминала (не в фоновом режиме);
* < – процесс поднял приоритет **(CPU scheduling priority);**
* E – процесс пытается завершиться;
* J – процесс выполняется в jail;
* L – некоторые страницы процесса заблокированы в памяти (например, непосредственный ввод-вывод);
* N – процесс уменьшил приоритет **(CPU scheduling priority);**
* s – этот процесс является лидером сеанса;
* V – родитель процесса приостанавливается во время vfork, ожидая завершения процесса или выхода;
* W – процесс выгружается;
* X – процесс отслеживается (trace) или отлаживается (debug).

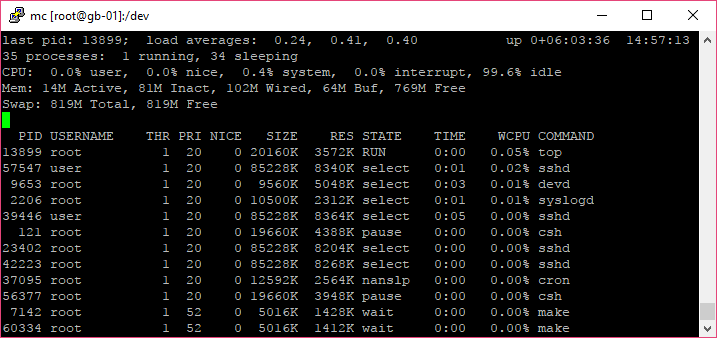
## top



Утилита, в реальном времени показывающая **Load Average** и информацию по процессам в интерактивном режиме. Отметим важное отличие top в Linux и FreeBSD. **Load Average** в FreeBSD показывает количество процессов, стоящих в очереди на обработку. Для многоядерных систем Load Average нужно делить на количество ядер. Так, если Load Average = 2 для 4-ядерной системы, мы имеем 2/4=0,5, что довольно неплохо. Считается, что если Load Average = 1 на ядро, это хорошо, иначе будут появляться очереди на обработку. В Linux Load Average – это не только процессы, стоящие в очереди на обработку, но и еще количество файлов в очереди. Поэтому в Linux по сравнению с аналогичной системой на FreeBSD Load Average окажется больше.

Ключ -P позволяет отобразить статистику по процессорно (per CPU usage statistics).

|  |
| --- |
| top -P |



## ktrace

Утилита ktrace похожа на strace в Linux: позволяет посмотреть системные вызовы, выполняемые отслеживаемым процессом. Утилита включает/выключает отслеживание событий ядра указанного процесса.

Пример: отслеживать все события ядра процесса с PID 49026.

|  |
| --- |
| ktrace -p 49026 |

Пример: прекратить отслеживать все события ядра для процесса с PID 49026.

|  |
| --- |
| ktrace -cp 49026 |

Пример: прекратить все отслеживания.

|  |
| --- |
| ktrace -C |

Результат трассировки записывается в файл ktrace.out. Он не человекочитаемый. Для анализа используйте утилиту kdump.

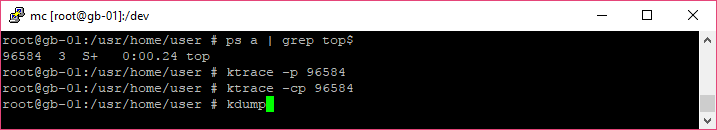
## kdump

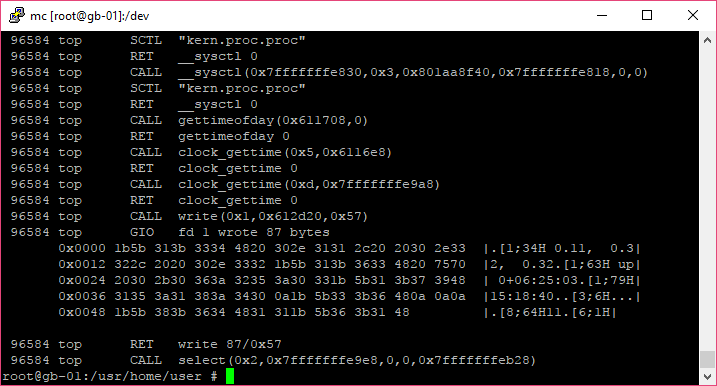
kdump служит для просмотра дампа отслеживания процесса, которое было включено утилитой ktrace.

Запуск:

|  |
| --- |
| kdump |

Пример:





## truss

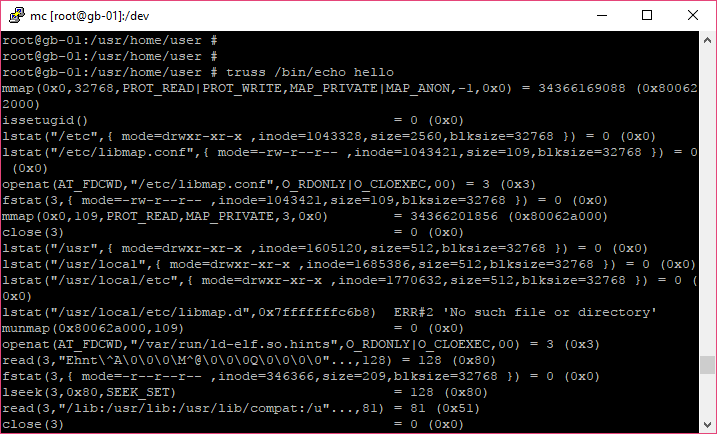
Также позволяет проанализировать системные вызовы.

Например:

|  |
| --- |
| truss |

или

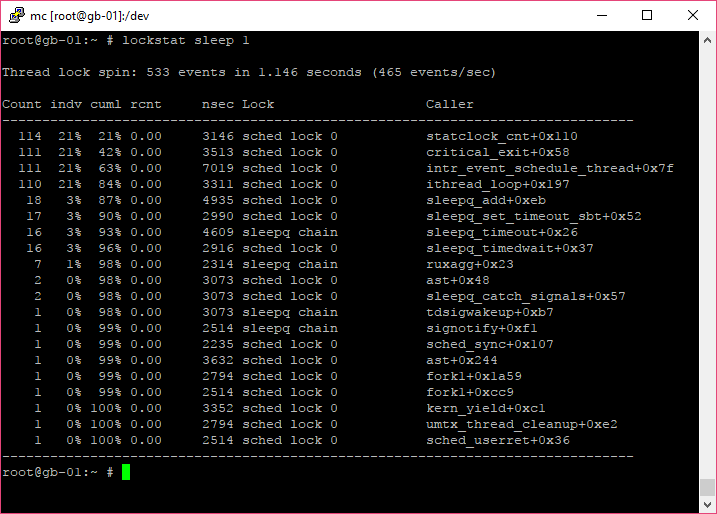
|  |
| --- |
| truss /bin/echo hello |



## lockstat

Позволяет посмотреть блокировки.

Пример:e



Подробнее:

|  |
| --- |
| man lockstat |

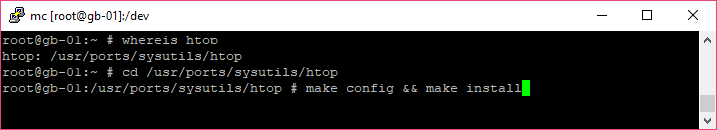
## htop

htop – интерактивный просмотрщик процессов. Портирован из linux, в настоящее время легко устанавливается и работает и во FreeBSD.

Можно установить из портов или с помощью pkg.

Установка из портов:

|  |
| --- |
| whereis htop  cd /usr/ports/sysutils/htop  make config  make install |

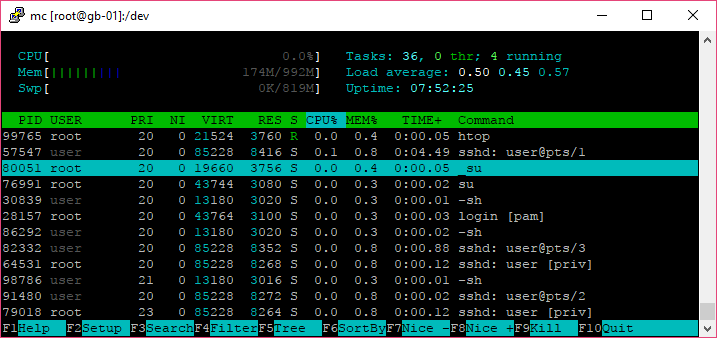


Установка с помощью pkg:

|  |
| --- |
| pkg install htop |

Запуск:

|  |
| --- |
| htop |



## DTrace

Утилита позволяет провести исследование, что происходит от уровня драйвера устройства (device driver) до уровня приложения (Application, см. иллюстрацию).

Утилита DTrace была разработана компанией Sun как инструмент для поиска узких мест в производственных и предпроизводственных системах. Это средство отладки и инструмент для анализа системы в реальном времени, локализации проблем производительности и других задач. DTrace – замечательный инструмент профилирования с внушительным набором возможностей для диагностики проблем системы

Позволяет проанализировать практически любое место в операционной системе.

# Сетевые утилиты

## netstat

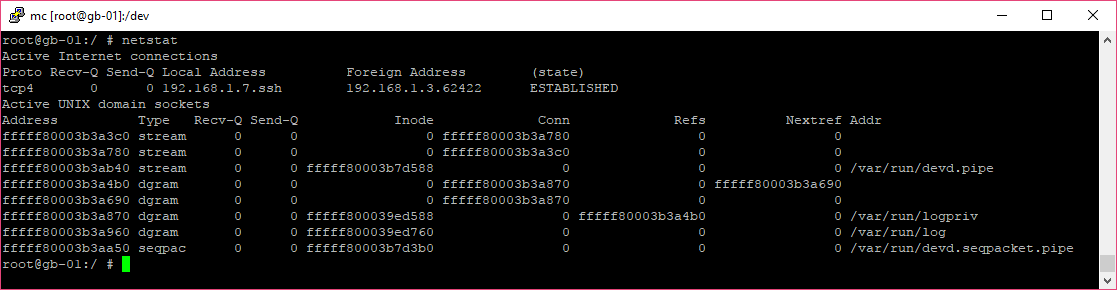
Позволяет выводить статистику по сетевым устройствам. Ключи отличны от Linux (команда netstat -ntl, которую часто используют, чтобы посмотреть прослушиваемые TCP-сокеты с номерами портов, работать не будет – не совпадают ключи, о чем нам сообщат сразу же).



Более подробно:

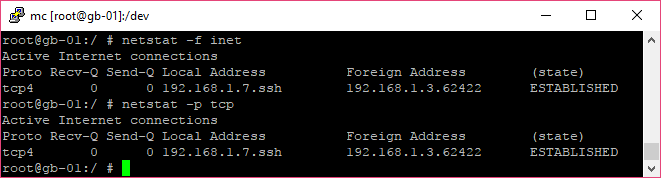
|  |
| --- |
| man netstat |

По умолчанию выдает статистику по активным подключениям.



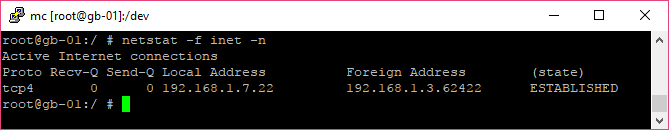
Здесь мы видим активное TCP-подключение (мы работаем через ssh) c указанием IP-адреса и порта на нашем хосте (в данном случае, куда подключено) и IP-адреса и порта удаленной машины (с которой мы подключились) и файловые сокеты UNIX-домена (stream, dgram и seqpac) с указанием имен файлов UNIX-сокетов (как правило в /var/run).

Примеры:



С помощью ключа -f мы указали семейство протоколов IPv4, с помощью ключа -p – протокол TCP.

Еще пример: с помощью ключа -n мы просим утилиту отображать номера портов, а не наименования стандартных для них протоколов.



## systat -ifstat

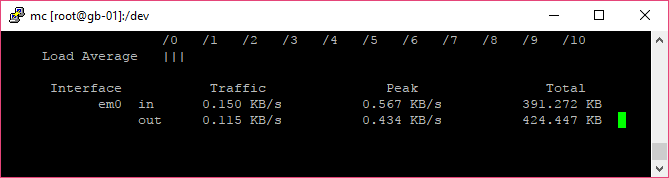
systat – утилита, позволяющая узнать информацию о многих компонентах системы.

Перечень ключей:

|  |
| --- |
| man systat |

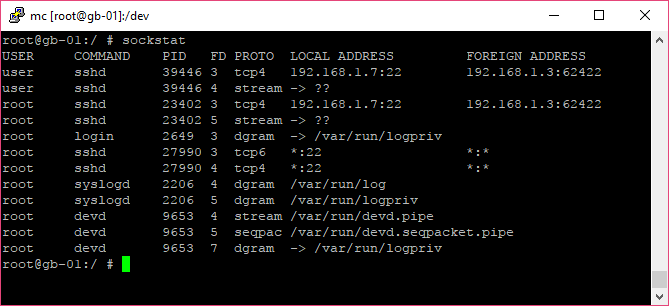
Например, загрузку сетевых интерфейсов можно посмотреть так:

|  |
| --- |
| systat -ifstat |

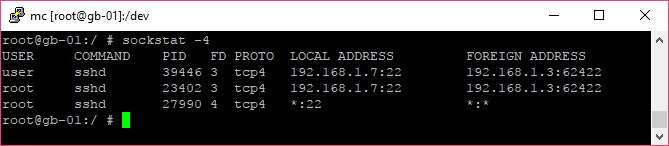


## sockstat

Вывод sockstat похож на netstat. Ключей у утилиты меньше, и она позволяет отобразить подключения в довольно удобном виде.



К примеру, если мы не используем ipv6, ключ -4 позволяет отобразить IPv4.

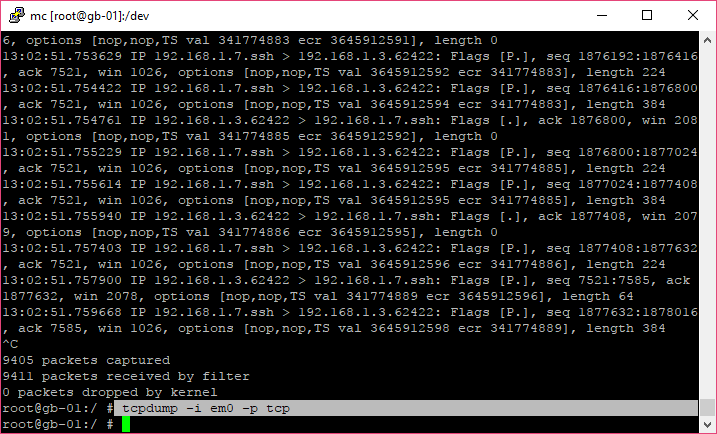


## tcpdump

Также во FreeBSD присутствует знакомая по ОС Linux утилита tcpdump – практически сниффер, который позволяет отслеживать не только свой, но и чужой трафик (для Wi-Fi или подключении с помощью концентратора, если сетевой интерфейс настроен в режиме promiscuous, то есть не отбрасывает фреймы, MAC-адрес получателя которых не совпадает с MAC-адресом устройства).

Пример: отследить на устройстве em0 весь tcp-трафик.

|  |
| --- |
| tcpdump -i em0 -p tcp |



Имеется возможность фильтровать по протоколам (IPv4, IPv6, TCP, UDP, ICMP и т. д.).

Подробнее:

|  |
| --- |
| man tcpdump |

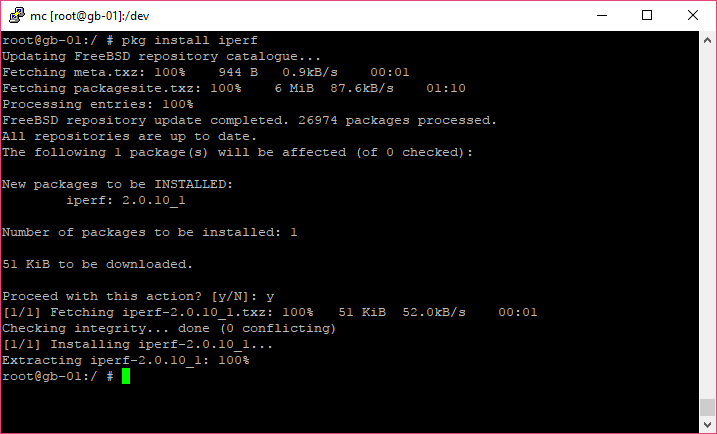
Утилита полезна для разрешения проблем в работе сетевых приложений, потому что позволяет отследить и проанализировать трафик на том или ином уровне стека TCP/IP.

## iperf

iperf – утилита, позволяющая замерить скорость между двумя точками сети. На одной стороне запускается сервер, на другой – клиент, после чего можно замерить пропускную способность канала.

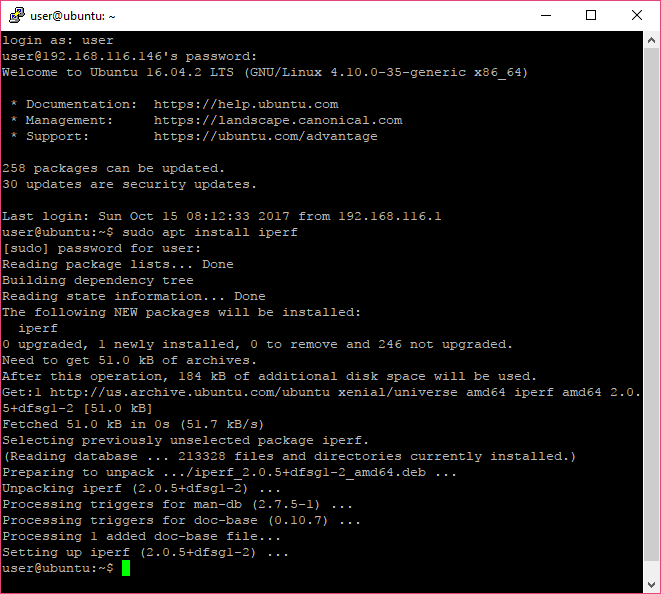
Установка:

|  |
| --- |
| pkg install iperf |

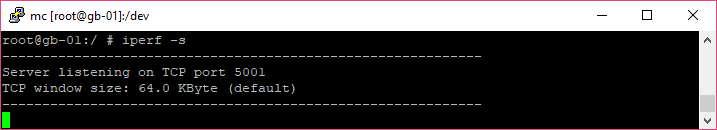


iperf имеется во всех UNIX-подобных операционных системах.

Например, в Ubuntu:

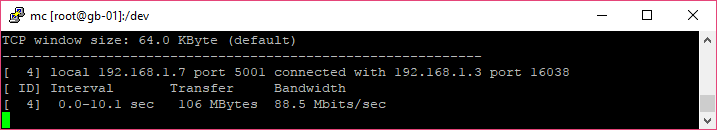


Далее запускаем в виртуальной машине с FreeBSD iperf в качестве сервера (используем ключ -s):



В виртуальной машине с Ubuntu запускаем iperf как клиент (используем ключ -c и указываем IP-адрес сервера):

На стороне сервера тоже видим статистику по скорости:



## mtr

mtr – my traceroute. Эта утилита есть также в Linux. Установим ее из портов.

Узнаем путь, где находится порт для mtr:

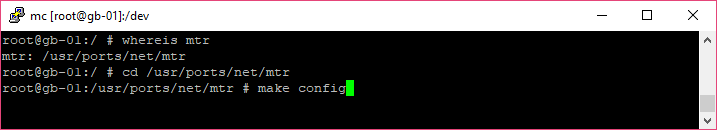
|  |
| --- |
| whereis mtr |

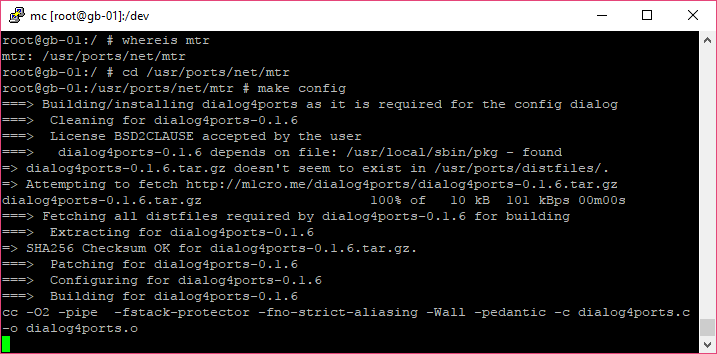
Перейдем в указанную директорию:

|  |
| --- |
| cd /usr/ports/net/mtr |

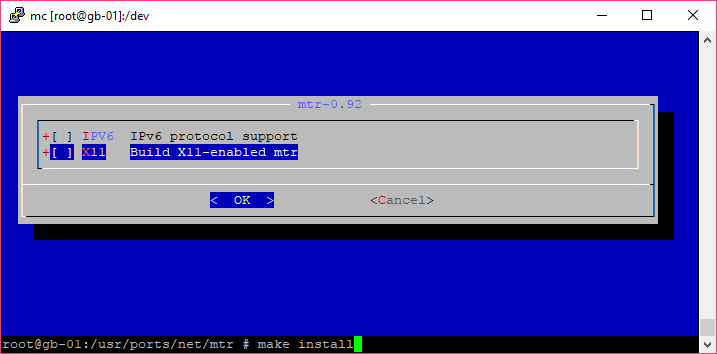
Выполним make config:

|  |
| --- |
| make config |

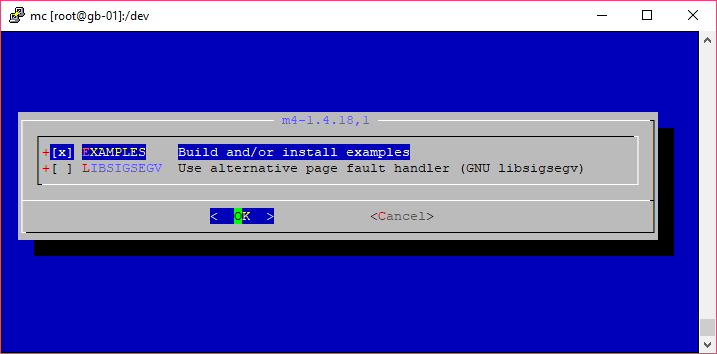




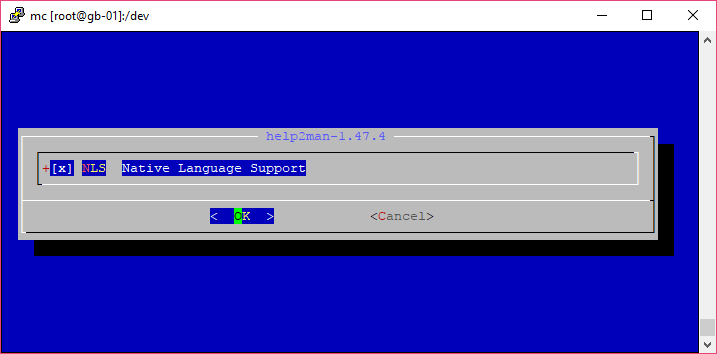
Отключим ненужное:

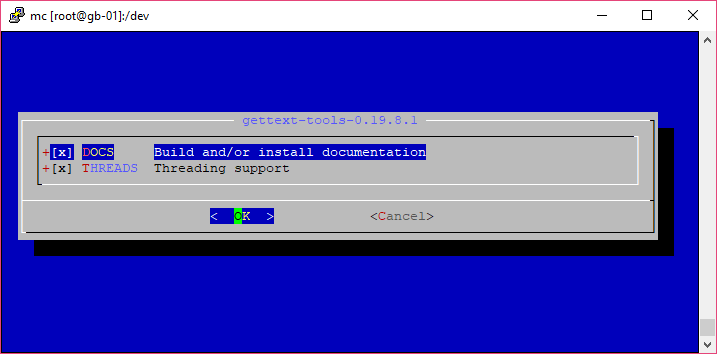


|  |
| --- |
| make install |



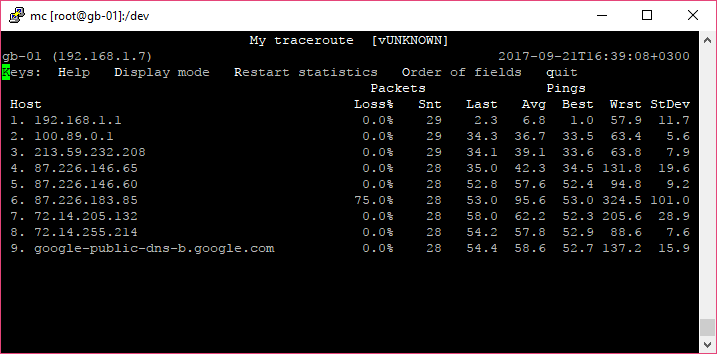
Если выберете поддержку языков, будут установлены все доступные языки.





Запуск:

|  |
| --- |
| mtr 8.8.4.4 |



Трассировка обновляется в реальном времени. Отображается процент потерь. Приемлемое значение потерь для сети – не более 10%.

# Установка ПО

## .Порты

Порты – один из способов распространения программного обеспечения во FreeBSD. Они выглядят как категории или рассортированные папки ПО, сценарии для установки.

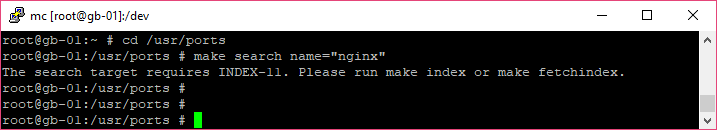
Порты хранятся в директории /usr/ports.

Поиск нужного ПО по дереву портов можно осуществлять с помощью whereis, если известно точное наименование утилиты.

С помощью make search программу можно найти, указав ее имя в значении параметра name=.

Сначала необходимо перейти в директорию /usr/ports и выполнить поиск:

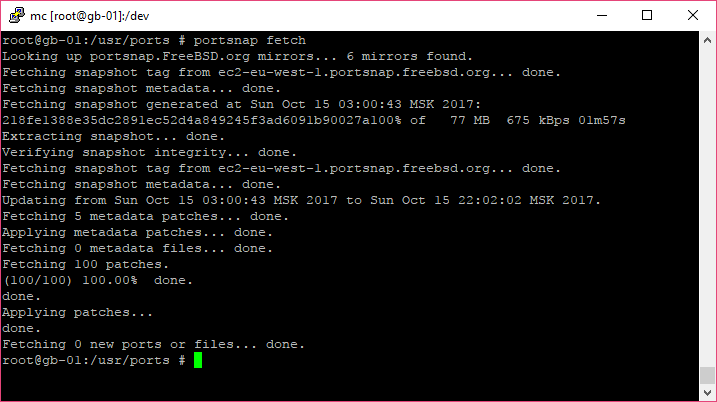
|  |
| --- |
| cd /usr/ports  make search name="nginx" |



Как мы видим, сначала необходимо обновить систему портов. Это можно сделать, например, следующим образом.

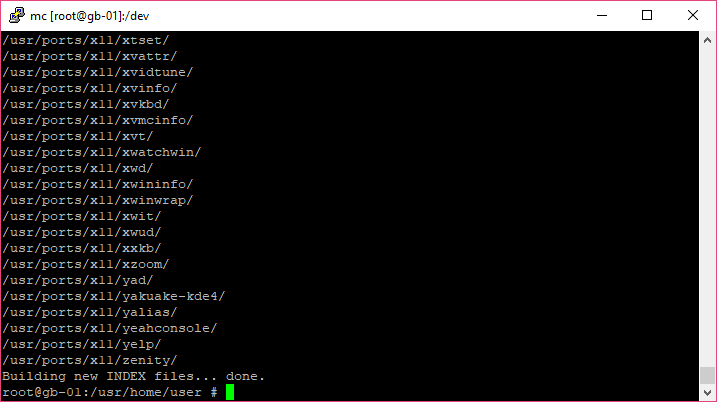
Сначала скачаем порты с помощью утилиты portsnap, запустив ее с ключом fetch:

|  |
| --- |
| portsnap fetch |



После того как все скачалось, распакуем:

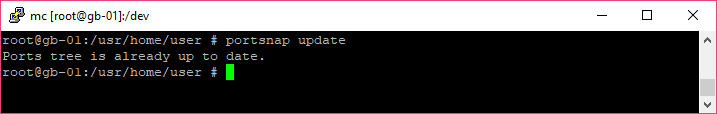
|  |
| --- |
| portsnap extract |
|  |



Индекс сформирован.

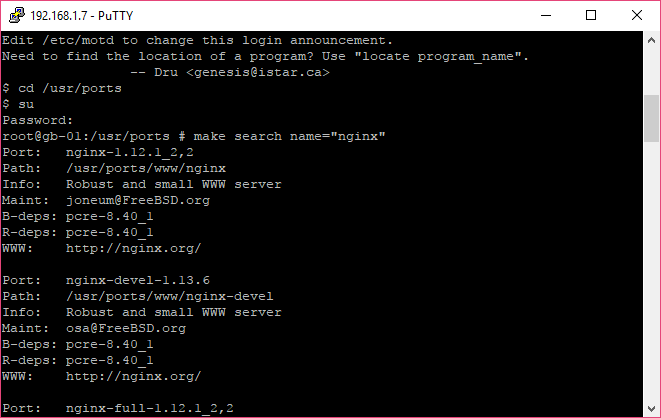
Удостоверимся, что все в полном порядке:

|  |
| --- |
| portsnap update |



Поиск. Используем name:top

|  |
| --- |
| cd /usr/ports  make search name="nginx" |
|  |



Обратите внимание, что выводятся доступные версии, включая девелоперскую, полную и т. д.

Если точное имя вы не знаете, вместо name можно использовать key. Выдача будет еще больше: туда добавятся вхождения указанного ключа.

|  |
| --- |
| make search key="nginx" |

После того как мы нашли порты, подходящие под критерий поиска, переходим в нужную директорию.

|  |
| --- |
| cd /usr/ports/www/nginx |

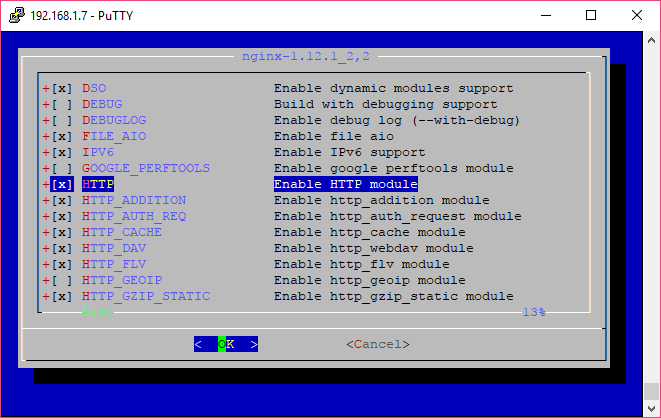
Посмотрим содержимое директории:

* Makefile – файл, с помощью которого мы будем собирать ПО. Содержит имя порта (PORTNAME), версию порта (PORTVERSION), MAINTAINER, опции.
* distinfo – содержит перечень архивов и контрольные суммы для проверки того, совпадают ли файлы с оригиналом.
* pkg-descr – description – короткое описание для пакета.
* pkg-plist – переменные, куда мы будем копировать сэмплы.
* директория files – патчи и дополнения для сборки.

После того как все скачивается, архивы складываются в /usr/ports/distfiles. Эту директорию периодически надо очищать. Также имеются пакеты с локализациями, по названиям языков видны в /usr/ports.

Сконфигурируем:

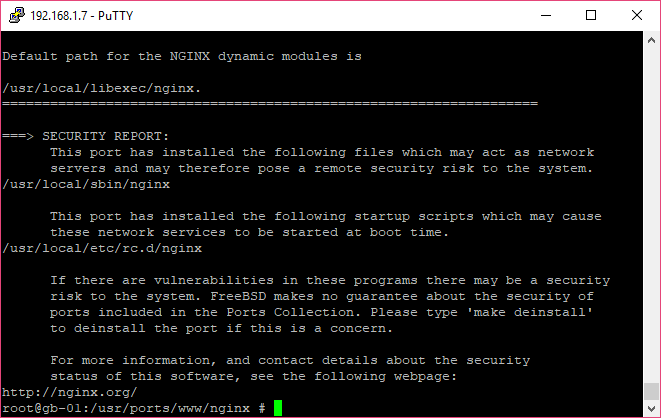
|  |
| --- |
| make config |
|  |



С помощью псевдографической среды можно выбрать необходимые модули, что очень удобно и практично.

Затем:

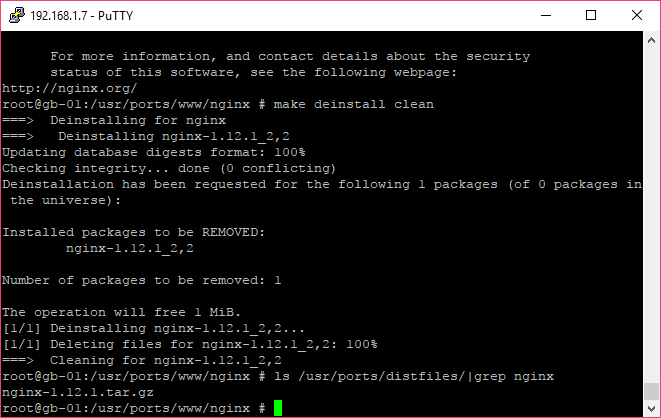
|  |
| --- |
| make install |



Если мы захотим удалить программу, используем deinstall.

|  |
| --- |
| make deinstall clean |

clean также очистит каталог от файлов, оставшихся после компиляции и сборки. Обратите внимание, что при этом архив по-прежнему остается в /usr/ports/distfiles.

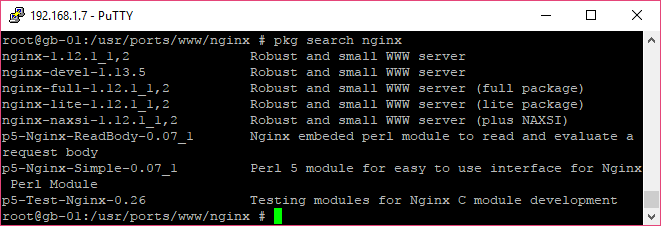


## pkg

В последнее время порты немного отходят на второй план. Они позволяют довольно точно настроить систему, но есть и недостаток: компиляция – это очень долго. Все чаще используется утилита pkg, похожая на apt из Debian/Ubuntu или yum из RedHat/Centos.

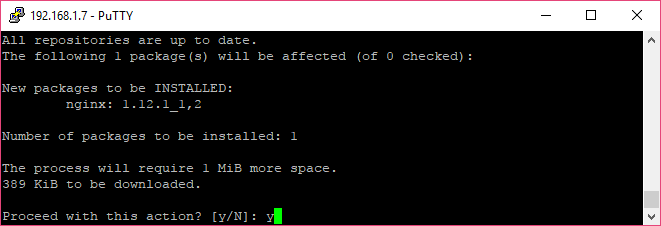
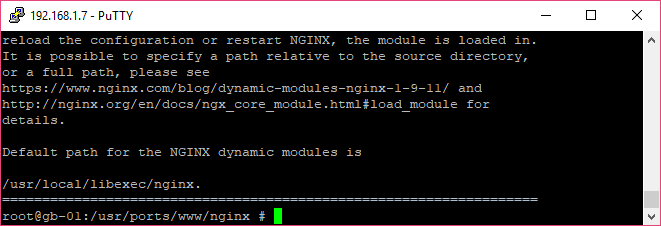
Поиск:

|  |
| --- |
| pkg search nginx |



Установка – c помощью install указываем имя пакета:

|  |
| --- |
| pkg install nginx |



Отвечаем утвердительно.

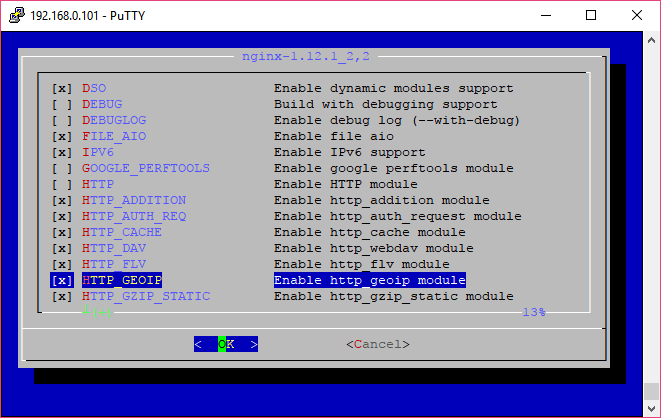
Сравните скорость pkg и портов.

## 

## Соберем пакет из порта на примере nginx

|  |
| --- |
| cd /usr/ports/www/nginx  make config |

Например, добавим GeoIP.

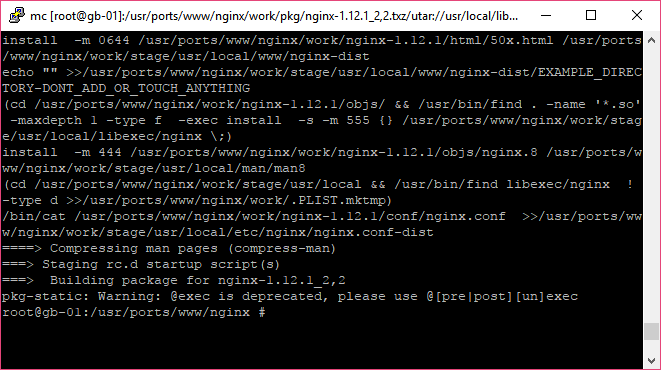


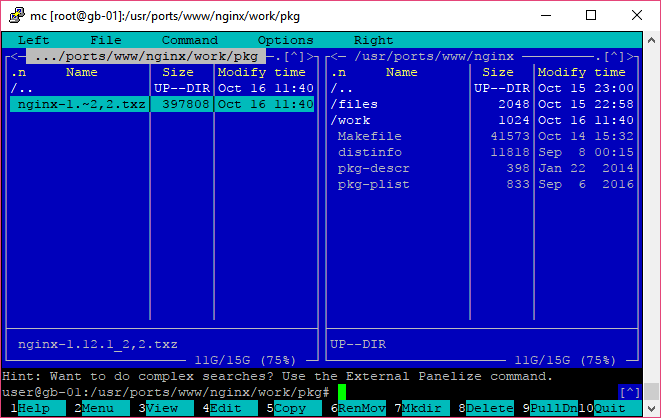
Добавляем модуль GeoIP, жмем **OK.**

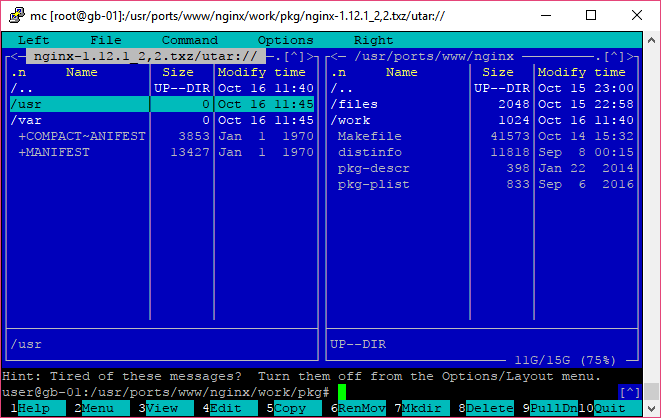
Далее при помощи команды

|  |
| --- |
| make package |

создается пакет. В данном случае мы его можем найти в /usr/ports/www/nginx/work/pkg.



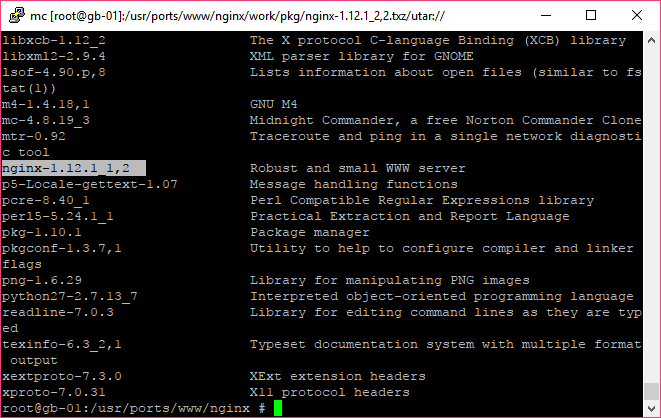
mc



Внутри +COMPACT-MANIFEST json с описанием манифеста (компактное, в +MANIFEST – полное. В /usr/local – бинарные файлы и необходимые скрипты, в /var – логи и tmp. Все это будет при установке скопировано в нужные директории.

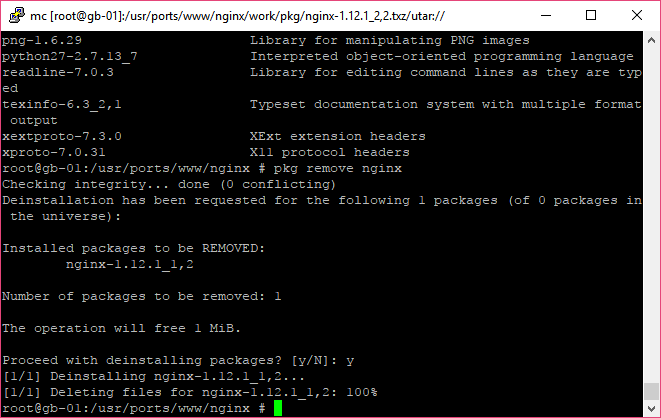
Посмотрим информацию об установленных пакетах:

|  |
| --- |
| pkg info |



Удалим nginx-1.12:

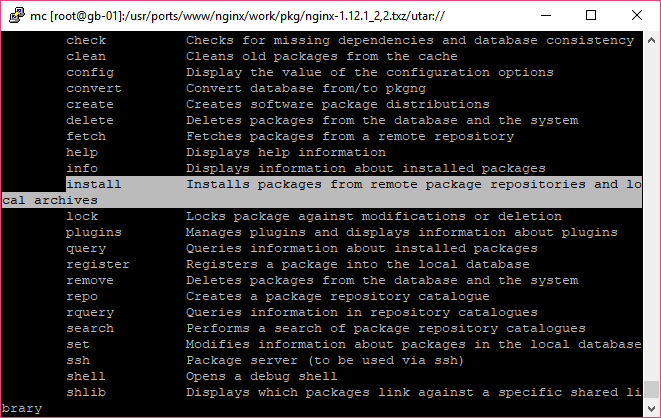
|  |
| --- |
| pkg remove nginx |



В pkg help

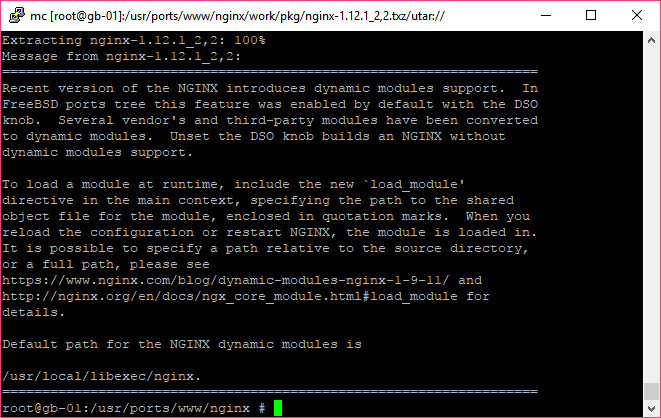
|  |
| --- |
| pkg help |

мы можем видеть, что install позволяет установить ПО из удаленного репозитория или локального архива



Установим из созданного нами пакета:

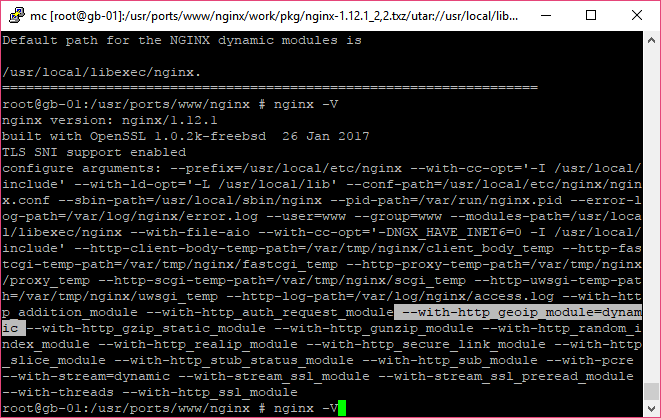
|  |
| --- |
| pkg install work/pkg/nginx-1.12.1\_2,2.txz |



С помощью

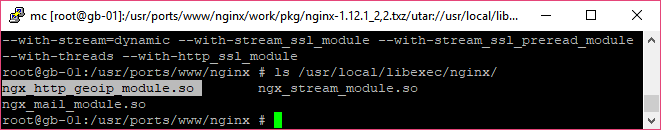
|  |
| --- |
| nginx -V |

проверьте список модулей.



В /usr/local/libexec/nginx, где хранятся динамически подключаемые модули nginx, видим добавленный модуль.

|  |
| --- |
| ls /usr/local/libexec/nginx |



Чтобы удалить зависимости, следует использовать pkg с ключом autoremove.

|  |
| --- |
| pkg autoremove |

Чтобы утилита не спрашивала подтверждений и не выводила сообщений в терминал (это можно делать через cron):

|  |
| --- |
| pkg autoremove -y -q |

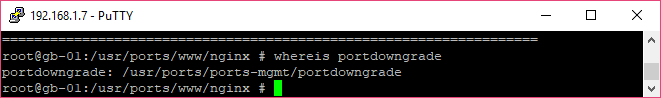
Если необходимо сделать пакет не из порта, а с нуля, есть команда

|  |
| --- |
| pkg config |

## portdowngrade

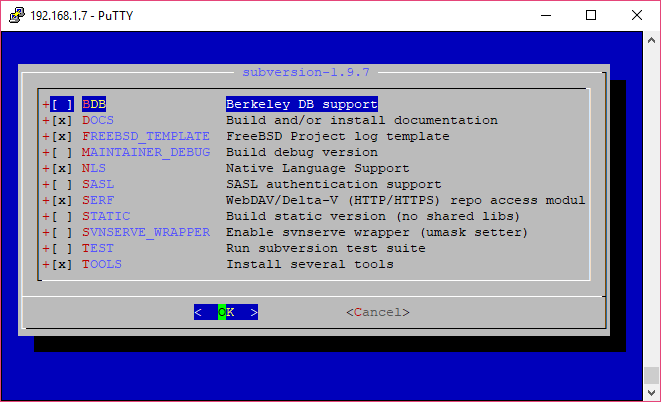
Утилита portdowngrade – это надстройка над портами. Она позволяет откатиться на любую версию пакета. Можно скачать порт для любой версии программного обеспечения.

|  |
| --- |
| whereis portdowngrade |

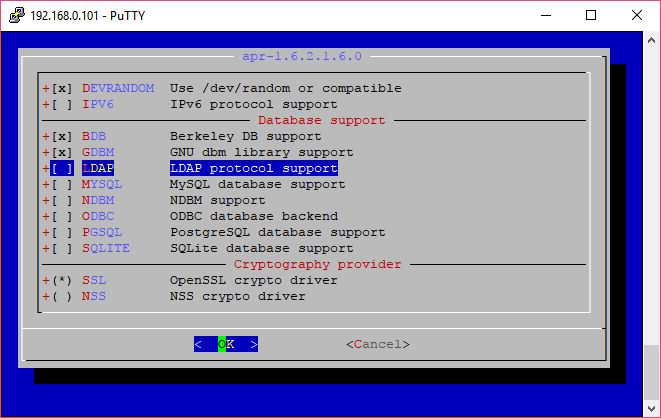


Для начала надо установить утилиту:

|  |
| --- |
| cd /usr/ports/ports-mgmt/portdowngrade  make install clean |



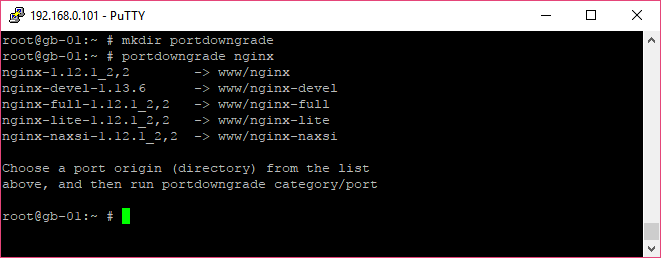
DOCS, NLS не нужны. Жмем **OK.**



Утилиту можно установить и с помощью pkg:

|  |
| --- |
| pkg install portdowngrade |

Использование: portdowngrade имя порта и ревизия.

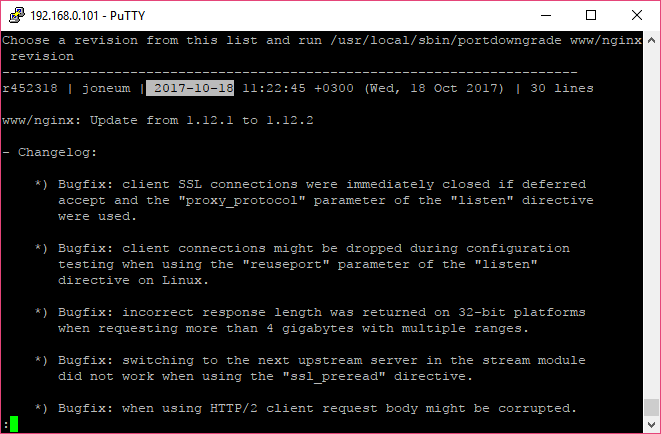


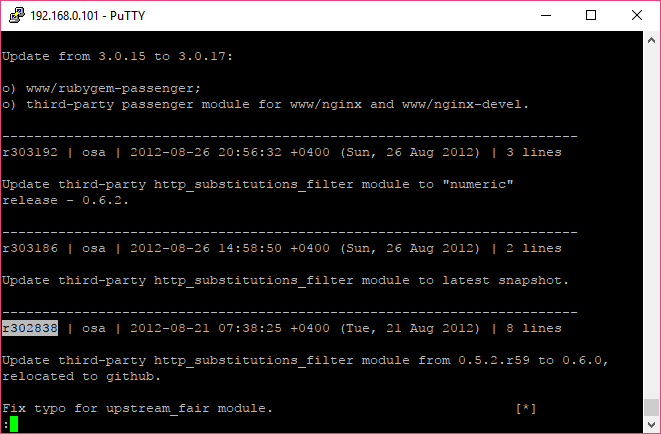
Например:

|  |
| --- |
| portdowngrade www/nginx |

Так как выдача займет несколько экранов, лучше сразу использовать less

|  |
| --- |
| portdowngrade www/nginx|less |

Видим номер ревизии, дату ревизии. Мы можем найти нужную ревизию:



и указать ее для portdowngrade.

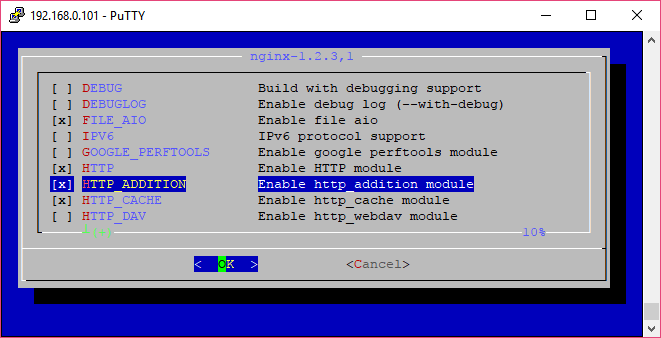
Переходим в созданную директорию portdowngrade и выполняем:

|  |
| --- |
| cd portdowngrade  portdowngrade www/nginx r302838 |

Мы получили директорию nginx.

Переходим в нее и выполняем make deinstall install clean:

|  |
| --- |
| cd nginx  make deinstall install clean |



Также можно использовать

|  |
| --- |
| make package |

Эта команда может быть полезна, если требуется использовать версию ПО, более раннюю, чем актуальная.

# Домашнее задание

1. Установить portdowngrade.
2. Изучить диаграмму с утилитами.
3. Собрать пакет для nginx, ntop, mtr.

# Дополнительные материалы

1. <https://www.freebsd.org/>
2. <https://www.freebsd.org/doc/handbook/>

# Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. <https://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/geom.html>
2. <http://phk.freebsd.dk/pubs/bsdcan-04.slides.geomtut.pdf>