Командный процессор Zsh

Alex Ott

8 января 2005 г.

Аннотация

Данная статья является кратким обзором возможностей коммандного процессора Zsh.

1 Введение

В этой статье предполагается, что читатель уже имеет навыки работы с одним из распространенных командных процессоров UNIX, и поэтому сосредоточимся на отличительных возможностях Zsh.

Zsh — командный процессор UNIX, который может использоваться как в качестве командного процессора для скриптов, так и в качестве интерактивной оболочки. Zsh обладает множеством возможностей, среди которых — настраиваемый механизм дополнений (completions), редактирование командной строки, механизм сохранения историй, встроенное исправление неправильных команд.

По возможностям Zsh наибольшее сходство имеет с Ksh, но при этом еще обладает множеством расширений.

Свежие версии Zsh доступны с сервера www.zsh.org[1] и его зеркал. Кроме этого, Zsh часто включают в состав популярных дистрибутивов Linux и других вариантов UNIX.

2 Запуск

Обычно Zsh указывают в качестве интерактивной оболочки для входа в систему. Но, кроме этого, Zsh можно запускать и отдельно, с разными ключами, определяющими его поведение. Например, ключ `-r` заставляет Zsh работать в «ограниченном»(restricted) режиме, ключ `-c` указывает откуда читать команды для выполнения, а ключ `-i` заставляет работать в интерактивном режиме.

При работе в «ограниченном»режиме запрещается выполнять некоторые действия: изменять каталог, запускать программы с помощью команды exec, перенаправлять вывод в файлы, изменять значение переменных среды, используемых при запуске программ, а также запускать программы, используя их абсолютные имена.

[1] http://www.zsh.org

При запуске Zsh старается эмулировать sh или Ksh в зависимости от того, под каким именем его запустили. В режиме эмуляции не исполняются обычные скрипты инициализации и завершения работы Zsh. Для инициализации используются файлы `/etc/profile` и `$HOME/.profile`

2.1 Файлы, используемые при запуске

При запуске Zsh читается некоторое количество файлов, содержащих команды инициализации. В первую очередь читается файл `/etc/zshenv`, затем читается файл `$ZDOTDIR/.zshenv`. Затем, если запускаемый процесс используется для входа в систему, то читаются файлы `/etc/zprofile` и `$ZDOTDIR/.zprofile`. Затем, если оболочка является интерактивной, читаются файлы `/etc/zshrc` и `$ZDOTDIR/.zshrc`. И наконец, если процесс используется для входа в систему, то читаются файлы `/etc/zlogin` и `$ZDOTDIR/.zlogin`.

Когда происходит выход из оболочки, использовавшейся для входа в систему, то читаются файлы `$ZDOTDIR/.zlogout` и затем `/etc/zlogout`.

Если переменная `ZDOTDIR` не установлена, то вместо нее используется значение переменной `HOME`. Файл `/etc/zshenv` читается при запуске любого процесса Zsh, поэтому он должен быть настолько малым, насколько это возможно.

Все перечисленные файлы могут быть откомпилированы с помощью команд **zcompile**, что позволяет их загружать быстрее. Откомпилированные файлы имеют расширение `.zwc`, и загружаются в том случае, если они новее исходного файла.

3 Грамматика команд

Грамматика команд по большей части совпадает с грамматикой других командных процессоров семейства bourne shell, хотя имеются различные расширения, специфические для данного командного процессора. Схожесть грамматики позволяет быстро перейти к работе с использованием Zsh, и осваивать новые возможности уже в процессе его эксплуатации.

4 Перенаправление вывода

Zsh поддерживает такое же перенаправление вывода как и Bash, но кроме этого он поддерживает множество расширений. Некоторые расширения работают только если заданы соответствующие настройки. Например, если не задана опция `CLOBBER`, то использование команды `>` для существующего файла приведет к возникновению ошибки и невыполнению команды. На этот случай, для практически всех команд существуют их модификации, которые не зависят от настроек. Например, для команды `>` существуют ее модификации `>!` и `>|`.

Для одновременного перенаправления стандартного вывода и стандартного потока сообщений об ошибках определены дополнительные команды `>&`, `&>`, `>>&`, `&>>` и их модификации со знаками `|` и `!` (например, `&>|`). Эти команды соответственно переписывают и дополняют информацию в указанных файлах, а также позволяют избавиться от неуклюжих конструкций вида `1>FNAME 2>&1`.

Zsh также позволяет перенаправлять информацию в более чем в один файл, аналогично использованию команды **tee** (это доступно только если определена переменная `MULTIOS`,

что обычно указано по умолчанию). Кроме этого можно одновременно использовать перенаправление и конвейеры. Например, команда **date** >`foo` | `cat` запишет дату в файл `foo` и выведет ее в стандартный поток вывода.

Если установлена переменная `MULTIOS`, то имя файла, указанное в качестве того, куда надо будет перенаправлять информацию, подвергается раскрытию шаблонов (globbing). Таким образом можно использовать команду

```
: > *
```

для усечения всех файлов в текущем каталоге.

Множественное перенаправление также работает и при вводе информации, позволяя заменять конструкции вида

```
cat foo fubar | sort
```

на конструкции

```
sort <foo <fubar
```

или даже

```
sort <f{oo,ubar}
```

Zsh даже позволяет использовать перенаправление без указания команды. Если не установлена переменная `NULLCMD`, то при таком использовании будет возникать ошибка. Но если установлена переменная `NULLCMD`, или `READNULLCMD` (она имеет преимущество над `NULLCMD`), то эти команды будут использоваться для обработки данных. По умолчанию `NULLCMD` имеет значением команду **cat**, а `READNULLCMD` значением команду **more**. Таким образом команда

```
< file
```

приведет к показу указанного файла.

5 Функции

Zsh позволяет пользователю определять собственные функции, которые могут выполняться точно также как и обычные программы. Функции выполняются в том же процессе, что и вызвавшая их программа. При вызове функции аргументы передаются как позиционные параметры.

Для ускорения загрузки Zsh может использовать автоматически загружаемые функции, когда они только объявляются, но не считываются. При первом использовании такой функции происходит ее считывание и компиляция. Для объявления автоматически загружаемой функции используется встроенная команда **autoload** (или ее налоги '`functions -u`' и '`typeset -fu`'). Поиск определений функций осуществляется по каталогам, перечисленным в переменной `fpath`.

Кроме обычных функций, Zsh позволяет определять функции со специальным значением. Так, например, функция **chpwd** вызывается при изменении рабочего каталога, а функции **precmd** и **preexec** выполняются перед каждым выводом приглашения и перед выполнением считанной команды соответственно. Кроме того, может быть определена функция **periodic**, которая может выполнятся через каждые `$PERIOD` (если этот параметр определен) секунд перед выводом приглашения.

6 Задания

Zsh работает с задачами подобно Bash, но при этом имеется возможность более гибко работать с заданиями запущенными в фоновом режиме. Кроме команды `&`, которая используется для запуска задачи в фоновом режиме, также определены команды `&|` или `&!`, которые запускают программу таким образом, что она не будет иметь записи в таблице задач и с которой нельзя будет работать обычными функциями работы с заданиями.

Для ссылки на задания можно использовать специальные переменные: `%NUMBER` — для ссылки на задание номер `NUMBER` (как в Bash); `%STRING` и `%?STRING` — для ссылки на задания, чьи командные строки начинаются и содержат строку `STRING` соответственно; `%%` (или `%+`) и `%-` для ссылки на текущее и предыдущее задание.

7 Подстановка переменных

Zsh имеет множество расширений в области различных подстановок. По назначению подстановки можно разделить на две группы: Подстановки в строке приглашения и прочие подстановки. Подстановки в строке приглашения могут использовать функции из группы «прочие подстановки».

7.1 Подстановки в строке приглашения

Zsh позволяет пользователю очень гибко настроить строки приглашения командного процессора. Для этого используются различные подстановки. С их помощью можно выделять цветами части текста в командной строке, помещать строку с текущим каталогом в строку окна терминала и многое другое.

Для приглашения командной строки существует разделение на левую и правую часть приглашения. Кроме привычных переменных, таких как `PS1` и `PS2`, также существуют их аналоги, имена которых начинаются с буквы `R` и которые определяют приглашения, выводимые в правой части экрана. Например, это позволяет вынести в правую часть часы, которые показывают время выведения приглашения.

Полное описание всех возможных подстановок для строки приглашения можно найти в справке по Zsh.

7.1.1 Темы для строки приглашения

Благодаря возможностям Zsh пользователи пишут различные модули расширений. Одним из таких модулей является модуль для установки тем для строк приглашения. Этот модуль загружается с помощью команд:

```
autoload -U promptinit
promptinit
```

и позволяет пользователям использовать заранее подготовленные темы приглашений. Команда **prompt -p** выдает примеры всех установленных тем строки приглашения.

7.2 Прочие подстановки

Все остальные подстановки выполняются в пять шагов, в перечисленном ниже порядке:

1. Сначала выполняются подстановки в истории. На этом этапе подставляются строки из истории, которые соответствуют заданным командам. См. раздел «XrefId[??]»П

2. Затем выполняется раскрытие псевдонимов (алиасов). Раскрытие алиасов выполняется немедленно, до того, как командная строка будет разобрана.

3. Затем выполняется подстановка процессов, параметров, команд, раскрытие арифметических выражений, и затем уже раскрытие выражений в скобках. Каждый из этих этапов подробно описан в справке по Zsh.

4. Раскрытие имен файлов. На этом этапе происходит подстановка комбинаций строк и чисел с символом ~. Если используется комбинация с числом, то каталог берется из стека каталогов. Если после символа ~ задается строка, то эта строка будет использоваться как имя пользователя, и вместо сочетания ~со строкой подставляется домашний каталог соответствующего пользователя.

5. И на последнем шаге происходит генерация имен файлов (globbing). См. раздел «XrefId[??]»

Полный перечень всех шаблонов и настроек вы сможете найти в справке по Zsh.

7.2.1 Подстановки истории

Система подстановок истории Zsh немного отличается от подстановок истории в командном процессоре bash. В Zsh сделано много добавлений, которые позволяют получить доступ к нужным аргументам выполненных команд, а также выполнить разные модификации аргументов.

Подстановки истории Zsh позволяют указать какое слово из команды нужно задействовать. Для этого используются следующие указатели слов:

`:0` первое слово в командной строке;

`:n` n-й аргумент;

`:^` 1-й аргумент (аналогично :1);

`:$` последний аргумент строки;

`:X-Y` аргументы с X по Y;

`:X-` все аргументы начиная с X кроме последнего;

`:*` все аргументы;

`:X*` все аргументы начиная с X (аналогично :X-$).

Кроме этого, в подстановках истории после указателей слов можно использовать модификаторы, которые позволяют выполнять различные подстановки в истории. Так, например, модификатор `:h` работает подобно использованию команды **dirname**, а модификатор `:t` работает подобно команде **basename**. Полный список модификаторов можно получить в руководстве по Zsh.

7.2.2 Генерация имен файлов

Zsh предоставляет множество дополнений и настроек по генерации имен файлов.

Операторы глоббинга Кроме стандартных символов (таких как `*`, `?`, `[...]`), используемых для генерации имен файлов в других командных процессорах, в Zsh определены дополнительные операторы глоббинга, такие как:

`^X` совпадает с любым кроме X. Например, `^*.elc` будет соответствовать всем файлам, кроме тех которые указаны в маске (в нашем случае это файлы с расширением `.elc`);

`X~Y` совпадает со всем, что соответствует шаблону X, и не соответствует шаблону Y. Например, `*.el*~*.elc` соответствует всем файлам, чье расширение начинается с `.el`, но при этом не будут учитываться файлы с расширением `*.elc`;

`X#` соответствует нулю или нескольким вхождениям шаблона X;

`X##` соответствует одному или нескольким вхождениям шаблона X.

Флаги глоббинга Флаги глоббинга используются для воздействия на шаблон, который стоит правее указанного флага. Все флаги имеют форму (`#X`), где `X` определяет нужный модификатор. Ниже приведено описание нескольких флагов, которые могут наиболее часто использоваться в работе:

`i` делает шаблон независимым от регистра символов, которые в нем применяются;

`l` заставляет символы в нижнем регистре, которые используются в шаблоне, соответствовать символам и в вернем и нижнем регистре. Символы в верхнем регистре, будут соответствовать точно тем же символам;

`I` локально отменяет воздействие флагов `l` и `i`;

другие флаги, описаны в руководстве по Zsh.

Квалификаторы глоббинга При генерации имен файлов могут использоваться квалификаторы глоббинга, так что пользователь может указывать какие типы файлов будут подпадать под под генерацию имен. Квалификаторы указываются в конце шаблона и заключаются в круглые скобки. Среди квалификаторов есть например,

`.` для указания обычных файлов,

`/` для указания каталогов,

`=` для указания сокетов,

и многие другие (можно указывать права доступа, времена модификации файлов и т.п.). Их полное описание можно найти в руководстве по Zsh.

Рекурсивный глоббинг Zsh позволяет производить рекурсивное раскрытие имен файлов. Для этого используются формы `**/` и `***/` Первая форма отличается от второй лишь тем, что не следует по символьным ссылкам. Так, например

```
ls **/foo
```

произведет поиск файлов с именем `foo` во всех подкаталогах.

Эти формы не могут комбинироваться с другими формами в одном шаблоне. При одновременном использовании данных форм с другими, оператор `*` приобретает свое обычное значение.

Кроме вышеперечисленных настроек. Zsh также позволяет использовать приблизительное соответствие, пытаясь найти файлы, в именах которых (по его мнению :-) могли бы быть сделаны ошибки.

8 Параметры и настройки

В Zsh различают параметры и настройки. Параметры используются для передачи и хранения данных, а настройки используются для управления поведением Zsh.

8.1 Параметры

Каждый параметр имеет имя, значение и набор атрибутов. Имя может состоять из букв, цифр и специальных знаков. Значениями могут быть числа (целые), строки, массивы и хеши (ассоциативные массивы).

Для объявления типов параметров или присвоения целого или строкового значения параметру, используется встроенная команда typeset. Значения строкового или целого типа могут присваиваться простым приравниванием — `ИМЯ=ЗНАЧЕНИЕ`. Для присваивания других типов параметров используется команда **typeset**.

Позиционные параметры используются для доступа к аргументам командной строки для функции или скрипта командного процессора. Специальные параметры `*`, `@` и `argv`

являются массивами, которые содержат все позиционные параметры. Позиционные параметры могут быть изменены после запуска скрипта или функции с помощью встроенной команды **set**.

8.2 Настройки

Настройки устанавливаются с помощью встроенной команды **setopt**, а удаляются с помощью команды **unsetopt**. Список настроек приведен в документации по Zsh.

9 Дополнения (completitions)

Дополнения обеспечивают набор строк за вас. Они могут быть в разных формах, но для их выполнения (обычно) используется клавиша **TAB**.

В Zsh дополняться может все что угодно: имена и пути файлов; имена встроенных и внешних команд, а также их ключи; имена переменных среды; имена пользователей и компьютеров, а также многое другое. Zsh имеет в своей поставке некоторое количество определений дополнений для основных программ операционных систем. Для первоначальной настройки дополнений, вам необходимо лишь вставить команду **compinit** в ваш файл инициализации, после чего Zsh сможет использовать дополнения для тех команд, настройка для которых имеется в поставке Zsh.

9.1 Виджеты дополнений

Виджеты дополнений определяются с помощью ключа `-C` встроенной команды **zle**, которая предоставляется модулем **zsh/zle**.

9.2 Дополнения с использованием compctl

Данный метод аналогичен по настройке тем дополнениям, что применяются в csh & tcsh и является устаревшим, так что новые пользователи скорее всего предпочтут использовать другие методы определения дополнений, но он все равно поддерживается текущими версиями Zsh.

Задание дополнений с помощью данного метода имеет общую форму:

```
compctl ключи [command ...]
```

Определение дополнения начинается с ключевого слова **compctl** и заканчивается списком команд для которых будет действовать дополнение.

Ключи определяют типы параметров, которые будут подпадать под дополнение (ключи командной строки, списки выполняемых заданий, и т.п.).

Ключ `-k` указывает, что параметры, которые будут дополняться, будут переданы в массиве. Это ключ очень полезен, когда у вас списки параметров приведены в файле, или заданы в явной форме. Так, можно явно задать список серверов, имена которых будут дополняться при использовании команды **ssh**:

```
compctl -k "( server1 server2 )" ssh
```

В том случае, если у вас много серверов, то вы можете перечислить их в файле и вместо явного перечисления указать в скобках команду для получения содержимого нужного файла (в нашем примере пусть это будет `~/.servers`):

```
compctl -k "( ‘ cat ~/.servers ‘ )" ssh
```

Ключ `-K` позволяет пользователю указать функцию, которая будет вызываться для получения списка дополняемых параметров.

Другие ключи команды **compctl** описаны в справочном руководстве по Zsh.

9.3 Подсистема дополнений

Эта подсистема является новой и введена начиная с версии 4.0. При этом для пользователей остается доступной старая подсистема дополнений с использованием команды **compctl**, однако можно запретить ее использование с помощью команды:

```
zstyle ':completion:*' use-compctl false
```

Основным отличием новой системы дополнений от старой является то, что вместо того, чтобы задавать все настройки разом при запуске командного процессора, соответствующие части кода будут вызываться только тогда, когда будет нажата клавиша **TAB**, при этом будут генерироваться новые дополнения. Также введены новые команды, которые заменяют использование команды **compctl** с разными ключами, например, для задания списка дополнений должна использоваться команда **compadd**.

Поведение команд зависит от используемого контекста, что позволяет генерировать разные списки дополнений для разных контекстов. Эта возможность, как и остальные, управляются с помощью команды **zstyle**. Например, можно выдавать список дополнений в виде меню, перебирать их циклически, а также построить небольшой менеджер файлов с помощью стандартных команд Zsh.

Полное описание всех возможностей новой системы дополнений можно найти в руководстве пользователя Zsh.

10 Модули Zsh

Часть возможностей Zsh оформлена в виде модулей, отделенных от основной части командного процессора. Каждый из этих модулей может быть подключен к командному процессору во время сборки, или может быть загружен динамически во время запуска.

Загрузка модулей производится с помощью команды **zmodload**. Например, для загрузки модуля редактирования командной строки **zle** будет использоваться команда

```
zmodload zsh/zle
```

10.1 Модуль редактирования командной строки (zsh/zle)

Данный модуль предоставляет пользователю возможность редактирования командной строки. Редактирование может осуществляться в одном из двух режимов — многострочном, если терминал поддерживает перемещение между строками и однострочном, если терминал не поддерживает перемещения между строками или установлена опция `SINGLE_LINE_ZLE`.

10.1.1 Привязки клавиш

Модуль редактирования поддерживает концепцию таблиц привязки клавиш (keymaps). В любой момент времени может существовать любое количество таблиц привязки. По умолчанию в Zsh определены 4 таблицы привязки клавиш, которые эмулируют команды Emacs, редактора vi в режиме вставки, редактора vi в режиме команд, а также специальный режим **safe**, которые не определяет никаких привязок, кроме вставки символов соответствующих клавишам.

10.1.2 Встроенные команды zle

Модуль **zle** предоставляет пользователю некоторое количество встроенных команд. Сюда относятся команды привязки клавиш — **bindkey**, которая позволяет определять и переопределять сочетания клавиш; команды управления виджетами **zle** — создания, удаления, выдачи списка и другие.

10.1.3 Виджеты zle

Все действия в модуле редактирования выполняются через виджеты. Задачей виджета является выполнение какой-то простой операции. Все клавиши привязываются к какому-то из виджетов. Виджеты могут быть встроенные или определенные пользователем.

Встроенные виджеты выполняют такие операции, как модификация текста, контроль истории, операции перемещения, работу с аргументами, контроль дополнений и другие. Поведение этих виджетов можно контролировать с помощью команд **zle** и **zstyle**.

Виджеты, определяемые пользователем, являются функциями командного процессора, которые могут выполнять любые команды. Кроме того, они могут запускать другие виджеты используя встроенные команды **zle**. Определяемые функции могут получать доступ к результатам выполнения команд, не создавая трудностей для пользователя.

10.2 Модуль zsh/zftp

Данный модуль позволяет пользователю выполнять команды **ftp** в командной строке или внутри скриптов. Интерфейс похож на тот, который используется в традиционном клиенте **ftp**, но при этом используются все возможности Zsh — дополнения, глоббинг и редактирование. Загрузка данного модуля предоставляет пользователю одну встроенную команду — **zftp**, с помощью которой и выполняются все операции.

Загрузка модуля выполняется с помощью команд:

```
autoload -U zfinit
zfinit
```

Встроенная команда **zftp** обеспечивает выполнение всех команд. Синтаксис команды выглядит следующим образом:

```
zftp подкоманда [аргументы]
```

Подкомандами являются обычные команды **ftp** — **open**, **cd**, **get**, **put**, **bin** и другие. Кроме того, модуль предоставляет другие команды, которые являются алиасами для команд «**zftp подкоманда**», такие как **zfopen**, **zfget**, **zfcd** и другие.

10.3 Другие модули Zsh

Кроме вышеперечисленных модулей, в поставку Zsh входят модули работы со списками привилегий POSIX.1e (модуль **zsh/cap**), работы с математическими функциями (модуль **zsh/mathfunc**), модуль использования некоторых стандартных команд (**chown**, **chgrp**, **ln**, **rm** и других) как встроенных (модуль **zsh/files**) и другие модули. Для получения их полного списка и описания работы с ними смотрите документацию из поставки Zsh.

11 Дополнительные источники информации

- Домашняя страница Zsh[2]
- Документация по Zsh[3]
- Руководство пользователя Zsh[4]
- Zsh FAQ[5]
- Zsh refcard[6]
- Zsh refcard на русском[7]

[2] http://www.zsh.org
[3] http://zsh.sunsite.dk/Doc/
[4] http://zsh.sunsite.dk/Guide/
[5] http://www.zsh.org/FAQ/
[6] http://zsh.sunsite.dk/Refcard/
[7] http://xtalk.msk.su/~ott/linux/zsh/zsh-refcard.pdf