Головы и теги

Дмитрий Халанский

21 сентября 2021 г.

Сохраним такой текст в файл `script.sh`:

```
cd /
```

Что произойдёт, если сделать `sh script.sh`?

Сохраним такой текст в файл `script.sh`:

```
cd /
```

Что произойдёт, если сделать `sh script.sh`? Ничего. Запустится новый процесс shell, у *нового* процесса сменится рабочая директория, затем исполнение завершится. Рабочая директория процесса, который выполнил `sh script.sh`, не изменится.

sourcing

Если сделать `. ./script.sh`, у *текущего* шелла сменится рабочая директория. Дело в том, что команда `.` означает “исполни текст в этом файле, не запуская новый shell”. Похоже на `#include` в Си: можно считать, что содержимое встраивается как есть.

Переменные окружения

При запуске в программу невидимым образом передаётся особый словарь, записи в котором — строки. В этом словаре обычно хранятся вещи, которые лень передавать как явные аргументы:

- Язык системы;
- Каким текстовым редактором пользоваться при нужде;
- Как называется shell, в котором всё выполняется;
- С какими опциями запускать Java...

Словарь можно менять изнутри программы.
См. `environ(7)` или `environ(3p)`.

Примеры программ

C, C++:

```c
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

extern char **environ;

int main()
{
        char** env = environ;
        while (*env) {
                char* pair = strdup(*env);
                const char* name = strtok(pair, "=");
                const char* value = strtok(NULL, "=");
                printf("name: %20s, value: %s\n", name, value);
                ++env;
                free(pair);
        }
        return 0;
}
```

Python:

```python
import os
print(os.environ)
```

Важные переменные

- `LANG` — язык системы. Если программа печатает результат в странной кодировке, попробуйте подать ей подходящий язык — или букву `C`, чтобы сказать ей “пиши как можешь, без изысков”.
- `PATH` — список директорий, в которых надо искать исполняемые файлы, через `:`.
- `HOME` — абсолютный путь к домашней директории.
- Много их. См. `https://pubs.opengroup.org/onlinepubs/007904875/basedefs/xbd_chap08.html` и `environ(7)`.

У разных программ есть важный для них набор переменных окружения. В man-страницах эта информация обычно в разделе `ENVIRONMENT VARIABLES`. Например, см. упоминание `VISUAL` в `git-commit(1)`.

Переменные окружения и shell

В shell есть обычные переменные (не переменные окружения). Присваиваются так (**без пробелов**):

```
$ my_var=3
```

Доступ к ним выглядит как $имя:

```
$ echo $my_var
```

Чтобы передавать значение переменной в другие программы как переменную окружения, надо выполнить команду

```
$ export my_var
```

Если у переменной то же имя, что у имеющейся переменной окружения, `export` можно не делать.

Переменные окружения и shell (2)

Можно передавать значения переменных окружения в программу, не используя обычные переменные shell:

```
$ my_var=3 python -c 'import␣os;␣print(os.environ["my_var"])'
3
```

Если хочется записать в переменную всё, что справа, то нужно заковать это в кавычки, как мы поступали с аргументами:

```
$ v="3␣python␣-c␣'import␣os;␣print(os.environ[\"v\"])'"
$ echo $v
3 python -c 'import␣os;␣print(os.environ["v"])'
```

Переменные окружения и shell (3)

Если в переменной есть пробелы, то при раскрытии она превратится в несколько слов:

```
$ x="echo␣␣␣5"
$ $x
5
$ python -c 'import␣sys;␣print(sys.argv)' $x
['-c', 'echo', '5']
```

Внутри двойных кавычек переменные раскрываются. Внутри одинарных — нет.

```
$ x="echo␣␣␣5"
$ python -c 'import␣sys;␣print(sys.argv)' "$x" '$x'
['-c', 'echo␣␣␣5', '$x']
```

dotfiles

Файлы конфигурации на Unix есть и хранятся обычно в таких местах:

- `/etc` — конфигурация, глобальная для системы;
- Среди скрытых файлов в домашней директории;
- В отдельной директории внутри `$HOME/.config`.

dotfiles ("файлы с точкой") — это общее название для файлов конфигурации. Их принято где-то аккуратно хранить, чтобы было легко воссоздать свою конфигурацию на новом компьютере.

rc-файлы

Частный случай файлов конфигурации — rc-файлы (run commands). В них содержатся команды, которые программа должна выполнить при запуске.

Примеры таких файлов:

- `$HOME/.bashrc` source-ится bash, если тот запущен как интерактивный shell (а не как интерпретатор), причём запущен не при логине пользователя в систему.
- `/etc/profile` и `$HOME/.bash_profile` (или `$HOME/.bash_login`, или `$HOME/.profile`) source-ятся bash, если тот запущен как интерактивный shell при входе пользователя в систему. См. `bash(1)`, раздел `INVOCATION`.
- `.muttrc` source-ится утилитой `mutt`.
- `.vimrc` source-ится текстовым редактором `vim`.

Головы

В `.git/refs/heads` лежат файлы-“головы”; в обиходе их называют “ветками”. Имя файла — название ветки. Содержимое файла — хэш коммита, на который указывает ветка; этот коммит называется “верхушкой” ветки (tip of the branch).
Вот и всё[1].

[1] Если ограничиваться локальным репозиторием, без сервера.

Теги

В `.git/refs/tags` лежат файлы-“теги” (“метки”). Имя файла — название тега. Содержимое файла — хэш помеченного (тегированного) коммита.

Например, принято помечать значимые состояния репозитория: “именно такой код был в репозитории, когда вышел релиз 2.0.0”.

HEAD

`.git/HEAD` (или просто `HEAD`) — это файл, в котором указан "головной коммит" — то, какое состояние репозитория выбрано сейчас. Там находится либо название ветки — тогда головной коммит является просто верхушкой ветки — либо хэш коммита — тогда говорят, что голова оторванная ("detached").
`HEAD` не может указывать на тег, только на ветку или коммит. Почему?

HEAD

`.git/HEAD` (или просто `HEAD`) — это файл, в котором указан "головной коммит" — то, какое состояние репозитория выбрано сейчас. Там находится либо название ветки — тогда головной коммит является просто верхушкой ветки — либо хэш коммита — тогда говорят, что голова оторванная ("detached").
`HEAD` не может указывать на тег, только на ветку или коммит. Почему? На следующем слайде.

Двигающие головной коммит операции

Как мы знаем, `git commit` не только создаёт новый коммит, но и делает его головным. На самом деле, если `HEAD` указывает на ветку, сам `HEAD` не меняется, а меняется содержимое ветки.
То же справедливо про `git reset`, `git cherry-pick`, `git rebase` и прочие операции, меняющие головной коммит.
Если бы `HEAD` мог указывать на тег, то эти операции меняли бы содержимое тега, а хочется, чтобы оно было замороженным.

Адресация коммитов

Многие операции в git требуют как аргумент какой-то коммит. В таких случаях можно подавать следующие вещи:

- Название ветки;
- Название тега;
- Хэш коммита;
- Префикс хэша коммита, если по этому префиксу нет коллизий;
- Всё перечисленное выше, к чему справа дописано n крышек $\hat{}$ — это состояние на n коммитов ранее.

Команды для голов

- `git branch` манипулирует ветками: создаёт, удаляет, переименовывает, перечисляет ветки.
- `git tag` делает то же с тегами.
- `git checkout BRANCH` устанавливает ГОЛОВУ на BRANCH и обновляет файлы в index и в рабочем дереве.
- `git checkout COMMIT` отрывает ГОЛОВУ, помещает её на COMMIT и обновляет файлы в index и в рабочем дереве.
- `git reset --soft COMMIT` переписывает ГОЛОВУ на COMMIT. Это всё равно что записать КОММИТ в `.git/refs/heads/текущая ветка`.
- `git reset --mixed COMMIT` переписывает ГОЛОВУ на COMMIT, а также меняет index на содержимое COMMIT.
- `git reset --hard COMMIT` переписывает ГОЛОВУ на COMMIT, а также меняет index и рабочее дерево на содержимое COMMIT.

git merge

`git merge COMMIT` — команда, которая ищет общего предка у ГОЛОВЫ и COMMIT и переводит ГОЛОВУ в состояние репозитория, в котором есть изменения как из головного коммита, так и из COMMIT, то есть осуществляет *слияние* двух наборов изменений. См `git-merge(1)`, там картинка.

Если COMMIT является потомком головного, то для слияния не нужно никаких изысков: сам COMMIT является результатом слияния, достаточно переписать ГОЛОВУ на коммит. См. раздел `FAST-FORWARD MERGE`.

Если же COMMIT не является потомком ГОЛОВЫ, происходит true merge.

git rebase

`git rebase` — команда, осуществляющая смену родительского коммита для набора изменений. Флаг `-i` кроет великую мощь.

git cherry-pick

`git cherry-pick` — команда, которая применяет наборы изменений из перечисленных коммитов к ГОЛОВЕ.