Начало работы с GIT

АВТОР: DWord ОПУБЛИКОВАНО 30.07.2012

Настройка работы GIT

```
# обязательные настройки
git config --global user.name 'Name Surname'
git config --global user.email 'mail@domain.com'

# чтобы в консоли отображался цвет
git config --global color.ui "auto"
git config --global color.status "auto"
git config --global color.branch "auto"
git config --global color.diff "auto"
git config --global color.interactive "auto"

git config --global core.editor vim

# usefull aliases
git config --global alias.ch checkout
git config --global alias.co commit
git config --global alias.st status
git config --global alias.br branch
git config --global alias.a add
git config --global alias.rsh "remote show"
git config --global alias.pl pull
git config --global alias.plo "pull origin"
git config --global alias.pu push
git config --global alias.puo "push origin"

# наглядная история коммитов по git hist
git config --global alias.hist "log --all --graph  --pretty=format:'%h - %an, %ar: %s'"

# более удобный diff
git config --global diff.wordregex '[[:alnum:]]+|[^[:space:]]'  # задает regexp по умолчанию для word-режима
git config --global alias.df diff --color-words=.
git config --global alias.dff diff --color-words
git config --global alias.dfl diff

# сокращения для командной оболочки
alias g='git'
alias got='git'
alias gti='git'
alias gir='git'
```

Пути к настройкам

- /etc/gitconfig
- ~/.gitconfig
- .git/config

Пример ~/.gitconfig

```
[user]
    name = I Am
    email = my@mail.work
[color]
    ui = auto
    status = auto
    branch = auto
    diff = auto
    interactive = auto
[alias]
    ch = checkout
    co = commit
    st = status
    br = branch
    df = diff
    a = add
    rsh = remote show
    pl = pull
    plo = pull origin
    pu = push
    puo = push origin
```

Настройка .gitignore

Это файл-список шаблонов игнорируемых файлов в рабочем каталоге. Может выглядеть, например так:

```
$ cat .gitignore
*.pyc
*.[oa]
*~
```

Настройка .gitattributes

Это файл, который определяет настройки, характерные для определенных путей в проекте.
Может быть размещен в .git/info/attributes, если нет желания отправлять его в репозиторий.

```
echo '*.pbxproj binary' >> .gitattributes  # трактовать файлы как бинарные и не diff'ить их

echo '*.doc diff=word' >> .gitattributes  # применять к файлам фильтр word при сравнении
# word - пользовательский фильтр, его надо задать
git config diff.word.textconv strings  # тут надо написать способ добычи текста из .doc

echo '*.png diff=exif' >> .gitattributes  # добавим фильтр для сравнения картинок
git config diff.exif.textconv exiftool  # по крайней мере будим diff'ать exif-данные
```

Графические клиенты для GIT'а

- `gitg` — `gtk`-клиент. Красивое дерево коммитов, возможность просматривать отдельные, только локальные или все ветви.
- `qgit` — `qt`-клиент. Менее красивое дерево коммитов, но `gitg` в случае сильно ветвящихся проектов делает неудобные переходы между некоторыми. Есть возможность просмотра всех или только локальных ветвей. Кроме всего позволяет искать по истории коммитов.

- `giggle` — еще один `gtk`-клиент, которые показывает дерево файлов для конкретного коммита. Можно отслеживать историю изменения одного файла.
- `tig` — консольный клиент, показывающий дерево коммитов. Умеет показывать все ветки или только локальные. Что еще от него нужно?

Создание локального репозитория

```
mkdir project
cd project
git init
git add .
git commit -m "Начальный коммит"
# working on
```

Копирование удаленного репозитория

```
git clone git://github.com/.../project.git folder_name
cd folder_name
# working on
```

Рабочий процесс

```
# edit files
vim main.py

# review changes
git status
git diff main.py

# stage the changes
git add main.py
git status

# commit the changes
git commit -m "Описание коммита"
```

Управление ветвями

```
git branch -a # список всех ветвей с указанием текущей
git branch feature  # создание новой ветви
git checkout feature  # переключение на ветвь
git branch -m feature new_feature  # переименование ветви
git branch -d new_feature  # удаление ветви
git merge feature  # слияние текущей и указанной ветвей
```

Работа с удаленным репозиторием

```
git remote show origin  # просмотр информации об удаленном репозитории
git push origin some_branch  # залить историю коммитов в удаленный репозиторий
git pull origin some_branch  # обновить историю коммитов из удаленного репозитория
```

Просмотр истории коммитов

```
git log [branch]
git log --pretty=oneline
```

Откаты изменений

```
git reset --hard  # полный сброс изменений до последнего коммита
git checkout myfile.txt  # сброс отдельного файла
git commit --amend  # переписывание лога сделанного коммита

git reset --soft HEAD^  # сделать исправления в последнем коммите
git add forgot.txt these.txt  # добавить, что забылось
git commit  # и снова записать
```