無料電子ブック

学習

Git

Free unaffiliated eBook created from
Stack Overflow contributors.

#git

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: git

It is an unofficial and free Git ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official Git.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

1: Gitをいめる

Gitはのバ―ジョンシステムであり、プログラマはので「スナップショット」コミットをしてコ―ドのをすることができます。コミットをすることで、プログラマはしいをでテスト、デバッグ、することができます。すべてのコミットは、コンピュータ、プライベ―トサ―バ、オ―プンソ―スのウェブサイトGithubなどでホストできる "Gitリポジトリ"としてられています。

Gitでは、コードのしい「ブランチbranch」をすることもできます。これにより、なるバ―ジョンのコードがいにんできることができます。これにより、1つのブランチにのバ―ジョンがまれ、のブランチにのがまれ、さらにのブランチになるセットがまれるシナリオがになります。 Gitはこれらのブランチをし、そのそれらをまとめて、ほとんどにするプロセスをります。

Gitはあなたのコ―ドに3つのなる ""をっています

- ワ―キングディレクトリ すべてのファイルの、、、およびをうです。
- ステ―ジング ディレクトリにえたをする
- リポジトリ Gitはプロジェクトのなるバ―ジョンとしてったをにします

Gitはもともとは Linuxカ―ネルソ―スをするためにられました。それらをよりにすることで、さなコミット、プロジェクトのフォ―ク、フォ―クのマ―ジ、のブランチをたくさんたせることができます。

CVSやSubversionにれているにとってののは、すべてのチェックアウトにソ―スツリ―だけでなく、プロジェクトのがまれていることです。リビジョンの、いリビジョンのチェック、ロ―カルへのコミット、ブランチの、のブランチのチェック、ブランチやパッチファイルのマ―ジなどのなは、すべてサ―バ―とすることなくロ―カルでできます。したがって、ちとののはりかれます。のをし、ロ―カルのをのにするには、「」リポジトリとのがです。これははなリポジトリをしているはでもをのにえましたあなたの「」はあなたがとするかをめるものです。

バ―ジョン

バ―ジョン	
2.13	2017-05-10
2.12	2017-02-24
2.11.1	2017-02-02
2.11	2016-11-29
2.10.2	2016-10-28
2.10	2016-09-02

バージョン	
2.9	2016613
2.8	2016-03-28
2.7	2015-10-04
2.6	2015-09-28
2.5	2015-07-27
2.4	2015-04-30
2.3	2015-02-05
2.2	2014-11-26
2.1	2014-08-16
2.0	2014-05-28
1.9	2014-02-14
1.8.3	2013-05-24
1.8	2012-10-21
1.7.10	2012-04-06
1.7	2010-02-13
1.6.5	2009-10-10
1.6.3	2009-05-07
1.6	2008-08-17
1.5.3	2007-09-02
1.5	2007-02-14
1.4	20060610
1.3	2006-04-18
1.2	2006-02-12
1.1	2006-01-08
1.0	20051221

バージョン	
0.99	2005-07-11

Examples

のリポジトリをし、ファイルをしてコミットする

コマンドラインでまずGitがインストールされていることをします

すべてのオペレーティングシステムで

```
git --version
```

UNIXのようなオペレーティングシステムの

```
which git
```

もされないか、コマンドがされないは、インストーラをダウンロードしてすることによってシステムにGitをインストールするがあります。にでなインストールについては、 Gitのホームページをしてください。

Gitをインストールしたら、ユーザとメールアドレスをします 。コミットするにこれをいます。

Gitがインストールされたら、バージョンにくディレクトリにし、のGitリポジトリをします

```
git init
```

これは、Gitがするためになをむしフォルダ`.git`します。

に、Gitがあなたのしいリポジトリにするファイルをします。このステップはながです。

```
git status
```

のファイルをします。 Gitにどのファイルをバージョンするかをえることができますパスワードなどのをむファイルやレポをさせるファイルのをける

```
git add <file/directory name #1> <file/directory name #2> < ... >
```

リストのすべてのファイルをリポジトリにアクセスできるすべてのとするがあるは、のディレクトリとそのサブディレクトリのすべてを1つのコマンドでします。

```
git add .
```

これは「のステージを」すべてのファイルがあなたののコミットにコミットするためにそれらを

し、バージョンにします。

バージョンのにならないファイルのは、 `add`コマンドをするに`.gitignore`というのファイルをしてします 。

コミットメッセージとともに、されたすべてのファイルをコミットします。

```
git commit -m "Initial commit"
```

これにより、されたメッセージでしいコミットがされます。コミットは、プロジェクトのまたはスナップショットのようなものです。これでリモートリポジトリにプッシュまたはアップロードすることができ、にじてでそれにることができます。
`-m`パラメータをすると、デフォルトのエディタがき、そこでコミットメッセージをしてすることができます。

リモートをする

しいリモートをするには、ターミナル、リポジトリがされているディレクトリの`git remote add`コマンドをします。

`git remote add`コマンドは2つのをとります

1. リモート `origin`
2. リモートURL`https://<your-git-service-address>/user/repo.git`

```
    git remote add origin https://<your-git-service-address>/owner/repository.git
```

リモートをするに、gitサービスでなリポジトリをするがあります。リモートをしたに、コミットをプッシュ/プルできるようになります。

リポジトリのクローン

`git clone`コマンドは、のGitリポジトリをサーバからローカルマシンにコピーするためにします。

たとえば、GitHubプロジェクトをクローンするには

```
cd <path where you'd like the clone to create a directory>
git clone https://github.com/username/projectname.git
```

BitBucketプロジェクトをするには

```
cd <path where you'd like the clone to create a directory>
git clone https://yourusername@bitbucket.org/username/projectname.git
```

これにより、ローカルマシンに`projectname`というディレクトリがされ、リモートGitリポジトリのすべてのファイルがされます。これには、プロジェクトのソースファイルと、プロジェクトのとをむ`.git`サブディレクトリがまれます。

`MyFolder`など、のディレクトリをするには

```
git clone https://github.com/username/projectname.git MyFolder
```

または、のディレクトリでする

```
git clone https://github.com/username/projectname.git .
```

1. されたディレクトリにする、ディレクトリはでもしなくてもかまいません。
2. のコマンドの`ssh`バ―ジョンをすることもでき`ssh`。

   ```
   git clone git@github.com:username/projectname.git
   ```

`https`バ―ジョンと`ssh`バ―ジョンはです。しかし、GitHubなどのホスティングサ―ビスのには、`ssh`ではなく`https`をすることをおめします。

アップストリ―ムリモ―トをする

フォ―クGithubのオ―プンソ―スプロジェクトをクロ―ンしているは、アップストリ―ムリポジトリへのプッシュアクセスがないがありますので、のフォ―クがですが、アップストリ―ムリポジトリをフェッチできるがあります。

にリモ―トをしてください

```
$ git remote -v
origin    https://github.com/myusername/repo.git (fetch)
origin    https://github.com/myusername/repo.git (push)
upstream  # this line may or may not be here
```

`upstream`がにあるそれはいくつかのバ―ジョンがあります、URLをするがありますはです

```
$ git remote set-url upstream https://github.com/projectusername/repo.git
```

アップストリ―ムがしない、または/のフォ―クをしたいはしません

```
$ git remote add upstream https://github.com/projectusername/repo.git
$ git remote add dave https://github.com/dave/repo.git
```

コ―ドの

あなたのコ―ドをするには、ロ―カルリポジトリをコピ―するリポジトリをリモ―トサ―バにします。

リモ―トサ―バ―のスペ―スのをにえるため、のリポジトリをします.1つは`.git`オブジェクトのみをち、ファイルシステムにコピ―をしません。ボ―ナスとして、 このリモ―トをアップストリ

—ムサ—バ—としてして、のプログラマ—とにアップデ—トをします。

リモ—トサ—バ—で

```
git init --bare /path/to/repo.git
```

ロ—カルマシン

```
git remote add origin ssh://username@server:/path/to/repo.git
```

`ssh:`はリモ—トリポジトリにアクセスするの1つにぎないことにしてください。

に、ロ—カルリポジトリをリモ—トにコピ—します。

```
git push --set-upstream origin master
```

`--set-upstream` または`-u` をすると、のないGitコマンドでされるトラッキングがされます `git pull` 。

ユ—ザ—と メ—ルの

```
You need to set who you are *before* creating any commit.  That will allow commits to have the
right author name and email associated to them.
```

リモ—トリポジトリにプッシュするにとはのもありません えば、GitHub、BitBucket、GitLabアカウントをしてリモ—トリポジトリにプッシュする

すべてのリポジトリのアイデンティティをするには、 `git config --global`います
これにより、ユ—ザ—の`.gitconfig`ファイルにがされます `$HOME/.gitconfig`は `%USERPROFILE%\.gitconfig`。

```
git config --global user.name "Your Name"
git config --global user.email mail@example.com
```

リポジトリのIDをするには、リポジトリで`git config`をします。
これは、々のリポジトリのを`$GIT_DIR/config`ファイルにします。 `/path/to/your/repo/.git/config`

```
cd /path/to/my/repo
git config user.name "Your Login At Work"
git config user.email mail_at_work@example.com
```

リポジトリのファイルにされているは、そのリポジトリをするときにグロ—バルよりもされます 。

ヒントなるアイデンティティオ—プンソ—スプロジェクト、、プライベ—トレポなどをっていて 、がしているのリポジトリごとにしいIDをするのをれたくはない

- グローバルIDをする

```
git config --global --remove-section user.name
git config --global --remove-section user.email
```

2.8

- グローバルではなく、リポジトリのでのみあなたのをすようにするには

```
  git config --global user.useConfigOnly true
```

そうすることで、えられたリポジトリにして`user.name`と`user.email`をしてコミットしようとするのをれると、のようにされます

```
no name was given and auto-detection is disabled
no email was given and auto-detection is disabled
```

コマンドについてぶ

gitコマンドの、つまりコマンドの、なオプション、そののドキュメントのについては、`--help` --オプションまたは`help`コマンドをしてください。

たとえば、 `git diff`コマンドにするなをすべてするには、のようにします。

```
git diff --help
git help diff
```

に、 `status`コマンドにするなすべてのをするには、のようにします。

```
git status --help
git help status
```

もされているコマンドラインフラグのをにするには、 `-h`します。

```
git checkout -h
```

SSH for Gitをセットアップする

Windowsをしているは、 Git Bashをきます 。 **Mac**または**Linux**をしているは、をきます。

SSHをするに、のSSHがあるかどうかをすることができます。

`~/.ssh`ディレクトリのをする

```
$ ls -al ~/.ssh
# Lists all the files in your ~/.ssh directory
```

されているSSHキーがにあるかどうかをするには、ディレクトリをしてください。デフォルトでは、のファイルはのいずれかです。

```
id_dsa.pub
id_ecdsa.pub
id_ed25519.pub
id_rsa.pub
```

Bitbucket、GitHubまたはのアカウントでしたいのとのペアがされているは、 `id_*.pub`ファイルのをコピーできます。

それのは、のコマンドでしいとのペアをできます。

```
$ ssh-keygen
```

EnterまたはReturnキーをして、デフォルトのをけれます。プロンプトがされたらパスフレーズをしてするか、のままにします。

SSHがssh-agentにされていることをしてください。バックグラウンドでしていないは、ssh-agentをバックグラウンドでします。

```
$ eval "$(ssh-agent -s)"
```

SSHをssh-agentにします。コマンドの`id_rsa`をファイルのできえるがあることにしてください。

```
$ ssh-add ~/.ssh/id_rsa
```

のリポジトリのをHTTPSからSSHにするは、のコマンドをします。

```
$ git remote set-url origin ssh://git@bitbucket.server.com:7999/projects/your_project.git
```

SSHでしいリポジトリをクローンするには、のコマンドをします

```
$ git clone ssh://git@bitbucket.server.com:7999/projects/your_project.git
```

Gitのインストール

Gitをってみましょう。まずに、インストールするがあります。あなたはそれをいくつかのでることができます。 2つのなものは、ソースからインストールするか、ごのプラットフォームののパッケージをインストールすることです。

ソースからのインストール

できれば、ソースからGitをインストールするのがにです。なぜなら、のバージョンをにれるからです。 Gitのバージョンにはなuiがみまれているので、ソースからソフトウェアをコンパイルするのがであるとじるは、バージョンをするのがのです。また、くのLinuxディストリビューションに

はにいパッケージがまれているもあります。にのディストリビューションやバックポートをしていないり、ソースからのインストールがのかもしれません。

Gitをインストールするには、curl、zlib、openssl、expat、およびlibiconvにするのライブラリがです。たとえば、yumFedoraなどまたはapt-getDebianベースのシステムなどをしているシステムをしているは、のコマンドのいずれかをしてすべてのをインストールできます。

```
$ yum install curl-devel expat-devel gettext-devel \
  openssl-devel zlib-devel

$ apt-get install libcurl4-gnutls-dev libexpat1-dev gettext \
  libz-dev libssl-dev
```

ながすべてったら、Git Webサイトからのスナップショットをすることができます

http://git-scm.com/downloadに、コンパイルしてインストールします。

```
$ tar -zxf git-1.7.2.2.tar.gz
$ cd git-1.7.2.2
$ make prefix=/usr/local all
$ sudo make prefix=/usr/local install
```

これがわれた、あなたはGitでアップデートのためにGitをすることもできます

```
$ git clone git://git.kernel.org/pub/scm/git/git.git
```

Linuxへのインストール

バイナリインストーラをってGitをLinuxにインストールしたいのであれば、にしているなパッケージツールをってGitをインストールすることができます。 Fedoraをっているなら、yumをうことができます

```
$ yum install git
```

あるいは、あなたがUbuntuのようなDebianベースのディストリビューションをっているなら、apt-getをしてみてください

```
$ apt-get install git
```

Macにインストールする

MacにGitをインストールするにはな3つのがあります。もなのはグラフィカルなGitインストーラをすることです。これはSourceForgeページからダウンロードできます。

http://sourceforge.net/projects/git-osx-installer/

1-7。 Git OS Xインストーラ。のなは、GitをMacPorts http://www.macports.orgでインストールすることです。 MacPortsがインストールされている、でGitをインストールしてください

```
$ sudo port install git +svn +doc +bash_completion +gitweb
```

すべてのエキストラをするはありませんが、Subversionリポジトリでgitをうがある8をには、+ svnをみむことをおめします。

Homebrew http://brew.sh/はGitをインストールするもうつのです。あなたがHomebrewをインストールしている、でGitをインストールしてください

```
$ brew install git
```

Windowsへのインストール

WindowsにGitをインストールするのはとてもです。 msysGitプロジェクトには、インストールがです。インストーラのexeファイルをGitHubページからダウンロードしてしてください

```
http://msysgit.github.io
```

それがインストールされた、コマンドラインのバージョンでになるSSHクライアントをむとのGUIのがあります。

*Windows*ののされているmsysGitシエルUnixスタイルをってGitをするがあります。このマニュアルではなコマンドラインをできます。なんらかので、ネイティブのWindowsシエル/コマンドラインコンソールをするがあるは、スペースをむパラメータののわりにをするがあります。また、サカンフレックスアクセントでわるパラメータをするがありますがWindowsのであるため、にがある

オンラインでGitをいめるをむ https://riptutorial.com/ja/git/topic/218/gitをいめる

2: .gitattributesファイルの

Examples

ラインをにする

`.gitattributes`ファイルをプロジェクトルートにします。

```
* -text
```

これは、 `core.autocrlf = false`をするのとじです。

ライン

`.gitattributes`ファイルをプロジェクトルートにします。

```
* text=auto
```

これにより、すべてのテキストファイルGitによってされるがLFでコミットされますが、ホストのオペレーティングシステムのデフォルトにってチェックアウトされます。

これは、される`core.autocrlf`デフォルトと`core.autocrlf`です。

- Linux / macOSでの`input`
- Windowsで`true`

バイナリファイルをする

Gitはバイナリファイルのにはかなりれていますが、バイナリファイルをにできます。
`.gitattributes`ファイルをプロジェクトルートにします。

```
*.png binary
```

`binary`は、 `-diff -merge -text`とのみみマクロです。

あらかじめめまれた.gitattributeテンプレート

`.gitattributes`ファイルにどのルールをリストするかわからないや、にけれられているをプロジェクトにしたいは、 `.gitattributes`ファイルをshooseまたはすることができます。

- https://gitattributes.io/
- https://github.com/alexkaratarakis/gitattributes

オンラインで.gitattributesファイルのをむ https://riptutorial.com/ja/git/topic/1269/-gitattributesファ

イルの

3: .mailmapファイルとメールののけ

- メールアドレスのみをきえる
  <primary@example.org> <alias@example.org>
- をメールアドレスできえる
  <primary@example.org>
- つのとメールでのエイリアスをマージする
  これは 'Other <alias2@example.org>'をけないことにしてください。
  <primary@example.org> <alias1@example.org><alias2@example.org>

`.mailmap`ファイルはのテキストエディタでで き、オプションのコントリビュータ、プライマリメールアドレス、およびエイリアスをむプレーンテキストファイルです。プロジェクトのルートの`.git`ディレクトリのにするがあります。

`git shortlog`や`git log --use-mailmap`ようなコマンドのなをするだけであることに`git log --use-mailmap`。これにより、コミットがきえられたり、さまざまなやメールアドレスでコミットされたりすることはありません 。

メールアドレスなどのについてコミットをするには、わりにgitフックをするがあります。

Examples

コミッターをエイリアスでマージして、ショートログにコミットをします。

がなるマシンやオペレーティングシステムのプロジェクトにすると、なるメールアドレスやがされ、リストやがすることがあります。

`git shortlog -sn`をしてのリストとそのコミットのをすると、のようながられます。

```
Patrick Rothfuss 871
Elizabeth Moon 762
E. Moon 184
Rothfuss, Patrick 90
```

このフラグメンテーション/ `.mailmap` 、メールマッピングをむプレーンテキストファイル`.mailmap`することでできます。

1にリストされているすべてのとメールアドレスは、それぞれのきエンティティにけられます。

のでは、マッピングはのようになります。

```
Patrick Rothfuss <fussy@kingkiller.com> Rothfuss, Patrick <fussy@kingkiller.com>
Elizabeth Moon <emoon@marines.mil> E. Moon <emoon@scifi.org>
```

このファイルがプロジェクトのルートにすると、`git shortlog -sn`すると、されたリストがされま

す

```
Patrick Rothfuss 961
Elizabeth Moon 946
```

オンラインで.mailmapファイルとメールののけをむ https://riptutorial.com/ja/git/topic/1270/-mailmapファイル-とメールののけ

4: Bash UbuntuのGitブランチ

き

このドキュメントでは、 **bash**のgitのブランチをいます。はgitブランチをにすがあります。のディレクトリへのパスとにブランチをすることができます。

Examples

ターミナルの

PS1とは

PS1はプロンプト1をします。これは、Linux / UNIXシェルでなプロンプトの1つです。をくと、bashプロンプトにPS1でされたがされます。 bashプロンプトにブランチをするには、PS1/.bash_profileのPS1のをするがあります。

gitブランチをする

/ .bash_profileにのをしてください

```
git_branch() {
 git branch 2> /dev/null | sed -e '/^[^*]/d' -e 's/* \(.*\)/ (\1)/'
}
export PS1="\u@\h \[\033[32m\]\w\[\033[33m\]\$(git_branch)\[\033[00m\] $ "
```

このgit_branchは、たちがいるブランチをつけます。このがわったら、のgit repoにされずにブランチをることができます。

オンラインでBash UbuntuのGitブランチをむ https://riptutorial.com/ja/git/topic/8320/bash-ubuntuのgitブランチ

5: Git Clean

- `git clean [-d] [-f] [-i] [-n] [-q] [-e <pattern>] [-x | -X] [--] <path>`

パラメーター

パラメータ	
-d	untrackedディレクトリにえて、untrackedディレクトリをします。トラッキングされていないディレクトリがのGitリポジトリによってされている、デフォルトではされません。このようなディレクトリをにしたいは、-fオプションを2してください。
-f、--force	Gitがきれいな`requireForce`がfalseにされていない、git cleanは-f、-nまたは-iをしないり、ファイルやディレクトリのをします。 Gitはの-fがされていないり.gitサブディレクトリまたはファイルをつディレクトリのをします。
-i、--interactive	でファイルのをします。
-n、--dry-run	にすることなく、するファイルののみがされます。
-q、 - quiet	にされたファイルのリストではなく、エラーのみをします。

Examples

されたファイルをする

```
git clean -fX
```

のディレクトリとすべてのサブディレクトリからすべてのされたファイルをします。

```
git clean -Xn
```

クリーニングされるすべてのファイルをプレビューします。

すべてのディレクトリをする

```
git clean -fd
```

されていないすべてのディレクトリとそののファイルがされます。のディレクトリからし、すべ

てのサブディレクトリをりしします。

```
git clean -dn
```

クリーニングされるすべてのディレクトリをプレビューします。

untrackedファイルをにする

```
git clean -f
```

untrackedファイルをすべてします。

インタラクティブなクリーン

```
git clean -i
```

するをし、のようなコマンドでをめます

```
Would remove the following items:
  folder/file1.py
  folder/file2.py
*** Commands ***
    1: clean        2: filter by pattern        3: select by numbers        4: ask each
    5: quit        6: help
What now>
```

のオプション`i`は、 `x` 、 `d`などののオプションとともにできます。

オンラインでGit Cleanをむ https://riptutorial.com/ja/git/topic/1254/git-clean

6: Git Diff

- `git diff [options] [<commit>] [--] [<path>…]`
- `git diff [options] --cached [<commit>] [--] [<path>…]`
- `git diff [options] <commit> <commit> [--] [<path>…]`
- `git diff [options] <blob> <blob>`
- `git diff [options] [--no-index] [--] <path> <path>`

パラメーター

パラメータ	
-p、-u、--patch	パッチをする
-s、--no-patch	diffをする。デフォルトでパッチをする`git show`ようなコマンドや、 - patchのをりすのに`--patch`
-	diffをのでする
--diff-algorithm=	アルゴリズムをします。はのとおりです `myers` 、 `minimal` 、 `patience` 、 `histogram`
-	、、モードなどのヘッダーのをする
--name-only	されたファイルののみをする
--name-status	されたファイルのとステータスをもなステータスはMみ、Aみ、Dみ
- チェック	によってマーカーまたはエラーがしたにします。エラーとみなされるものは、`core.whitespace`によってされます。デフォルトでは、のだけでされるもむとののインデントのにタブがくは、エラーとみなされます。がつかった、ゼロのでします。 --exit-codeとがありません
--full-index	のいくつかののわりに、パッチのをするときに、ののBLOBオブジェクトを「インデックス」にします
- バイナリ	`--full-index`にえて、 `git apply`できるバイナリdiffをし`git apply`
-	すべてのファイルをテキストとしています。
-	カラーモードをします。 diffをlessにパイプしたいは`--color=always`い、gitのづ

パラメータ	
	けをつ

Examples

ブランチのいをする

```
git diff
```

これは、のコミットからののブランチのステージングされていないをします。インデックスにしたのみがされます。つまり、のコミットにできるがされますが 、はされません。これらのをステージングするには、 `git add`をします。

ファイルがステージングされているが、ステージングにされた、 `git diff`はのファイルとステージングされたバージョンのいをします。

ステージングされたファイルのをする

```
git diff --staged
```

これにより、のコミットとステージングされているファイルののがされます。

のコマンドをしてじことをすることもできます。

```
git diff --cached
```

`--staged`または`--staged`なるです

```
git status -v
```

`status`コマンドのながトリガーされます。

ステージングされたとステージングされないのをする

すべてのステージングされたおよびステージングされないをするには、

```
git diff HEAD
```

のコマンドをすることもできます。

```
git status -vv
```

は、のがにどのがコミットのためにステージングされているのか、そうではないのかをえること

です。

2つのコミットののをする

```
git diff 1234abc..6789def    # old   new
```

の3つのコミットにえられたをする

```
git diff @~3..@    # HEAD -3   HEAD
```

2つのドット..はオプションですが、さをします。

これは、ツリ―のどこにあるかにかかわらず、コミットのテキストのいをします。

meldをしてディレクトリのすべてのをする

```
git difftool -t meld --dir-diff
```

ディレクトリのがされます。あるいは、

```
git difftool -t meld --dir-diff  [COMMIT_A] [COMMIT_B]
```

2つののコミットのいをします。

のファイルまたはディレクトリのをする

```
git diff myfile.txt
```

したファイル `myfile.txt` ののコミットと、まだステ―ジングされていないロ―カルでされたバ―ジョンとののをします。

これはディレクトリでもします

```
git diff documentation
```

は、されたディレクトリ `documentation/` のすべてのファイルののコミットと、まだステ―ジングされていないこれらのファイルのロ―カルにされたバ―ジョンとののをしています。

のコミットのファイルのあるバ―ジョンとロ―カルの`HEAD`バ―ジョンとのいをするには、したいコミットをすることができます

```
git diff 27fa75e myfile.txt
```

または、2つのしたコミットののバ―ジョンをするは、のようにします。

```
git diff 27fa75e ada9b57 myfile.txt
```

ハッシュ`ada9b57`されたバージョンと、ブランチ`my_branchname`のコミットを、 `my_branchname`というディレクトリのみとの`my_changed_directory/`をするには、のようにします。

```
git diff ada9b57 my_branchname my_changed_directory/
```

いのためのの

```
git diff [HEAD|--staged...] --word-diff
```

されたをするのではなく、のをします。たとえば、のようになります。

```
-Hello world
+Hello world!
```

がみとしてマークされている、 `word-diff`はをのようにします。

```
Hello [-world-]{+world!+}
```

`--word-diff=color`または`--color-words`すると、マーカー`[-` 、 `-]` 、 `{+` 、 `+}`をでき`--color-words`。これはけをしてをマークします

```
@@ -1 +1 @@
Hello worldworld!
```

のをむ3マージの

```
git config --global merge.conflictstyle diff3
```

コンフリクトしたセクションののフォーマットではなく、 `diff3`スタイルをデフォルトとしてし、2つのファイルをします。

```
<<<<<<< HEAD
left
=======
right
>>>>>>> master
```

のテキストのからるをむセクションがまれます。

```
<<<<<<< HEAD
first
second
|||||||
first
=======
last
```

```
>>>>>>> master
```

このでは、マ―ジのをにします。この、ロ―カルに`second`がされ、リモ―トは`first`から`last`にされ、のようにされます。

```
last
second
```

をすると、じがはるかにしくなります。

```
<<<<<<< HEAD
first
second
=======
last
>>>>>>> master
```

のバ―ジョンとのバ―ジョンのいをする

```
git diff HEAD^ HEAD
```

これにより、のコミットとのコミットののがされます。

Diff UTF-16でエンコ―ドされたテキストおよびバイナリplistファイル

これらのファイルをどのようにdiffするべきかをすることで、UTF-16でエンコ―ドされたファイルロ―カライゼ―ションファイルoos iOSとmacOSはですをできます。

`~/.gitconfig`ファイルにをしてください。

```
[diff "utf16"]
textconv = "iconv -f utf-16 -t utf-8"
```

`iconv`はなるエンコ―ディングをするプログラムです。

に、するリポジトリのルートに`.gitattributes`ファイルをまたはします。または、`~/.gitattributes`するだけ`~/.gitattributes`。

```
*.strings diff=utf16
```

git `.strings`に`.strings`わるすべてのファイルをします。

テキストにできるのファイルについてものことができます。

バイナリplistファイルのは`.gitconfig`をします`.gitconfig`

```
[diff "plist"]
```

```
textconv = plutil -convert xml1 -o -
```

と.gitattributes

```
*.plist diff=plist
```

ブランチの

newのと**original**ののののをす

```
git diff original new     # equivalent to original..new
```

originalからして、すべてのを**new**する

```
git diff original...new     # equivalent to $(git merge-base original new)..new
```

1つのパラメ―タのみをする

git diffオリジナル

は

git diff original ..HEAD

2つのブランチでをする

```
git diff branch1..branch2
```

パッチの**diff**をする

によっては、patchをしてするdiffがなもあります。の`git --diff`はしません。わりにこれをしてください

```
git diff --no-prefix > some_file.patch
```

に、あなたはそれをにすことができます

```
patch -p0 < some_file.patch
```

2つのコミットまたはのい

2つのブランチのいをするには

```
git diff <branch1>..<branch2>
```

2つのブランチのいをするには

```
git diff <commitId1>..<commitId2>
```

のブランチでdiffをするには

```
git diff <branch/commitId>
```

のをするには

```
git diff --stat <branch/commitId>
```

のコミットにされたファイルをするには

```
git diff --name-only <commitId>
```

ブランチとはなるファイルをするには

```
git diff --name-only <branchName>
```

のコミットにフォルダでされたファイルをするには

```
git diff --name-only <commitId> <folder_path>
```

オンラインでGit Diffをむ https://riptutorial.com/ja/git/topic/273/git-diff

7: Git Remote

- `git remote [-v | --verbose]`
- `git remote add [-t <branch>] [-m <master>] [-f] [--[no-]tags] [--mirror=<fetch|push>]<name> <url>`
- `git remote rename <old> <new>`
- `git remote remove <name>`
- `git remote set-head <name> (-a | --auto | -d | --delete | <branch>)`
- `git remote set-branches [--add] <name> <branch>...`
- `git remote set-url [--push] <name> <newurl> [<oldurl>]`
- `git remote set-url --add [--push] <name> <newurl>`
- `git remote set-url --delete [--push] <name> <url>`
- `git remote [-v | --verbose] show [-n] <name>...`
- `git remote prune [-n | --dry-run] <name>...`
- `git remote [-v | --verbose] update [-p | --prune] [(<group> | <remote>)...]`
- `git remote show <name>`

パラメーター

パラメータ	
-v、--verbose	にします。
-m <master>	headをリモートの<master>ブランチにする
--mirror = fetch	Refsはrefs / remotesにされず、ローカルのrepoにミラーリングされます
--mirror = push	`git push`は--mirrorがされたかのようにします
--no-tags	`git fetch <name>`はリモートリポジトリからタグをインポートしません
-t <ブランチ>	<ブランチ> のみをするリモートをします。
-f	`git fetch <name>`は、remoteがされたにされます
- タグ	`git fetch <name>`はリモートのリポジトリからすべてのタグをインポートする
-a、--auto	symbolic-refのHEADは、リモートのHEADとじにされています
-d、--delete	されたすべてのは、リモートリポジトリからされます
--add	されているブランチのリストに<name>をしますset-branches
--add	いくつかのURLをするわりに、しいURLがされますset-url
- すべて	すべてのをしてください。

パラメータ	
-	<url>とするすべてのURLがされます。 set-url
- す	URLをするわりにプッシュURLがされます
-n	リモートヘッドは`git ls-remote <name>`にされず、キャッシュされたがわりにされます
--dry-run	どのがされるのかをするが、にはしない
- プルーン	ローカルのがないリモートブランチをする

Examples

リモートリポジトリをする

`git remote add`には、ローカルリポジトリのルートに`git remote add`をします。

ない<name>としてリモートのGitリポジトリ<url>をするには

```
git remote add <name> <url>
```

コマンド`git fetch <name>`をして、リモートトラッキングブランチ`<name>/<branch>`をしてすることができます。

リモートリポジトリのをする

`<old>`というのリモートのを`<new>`ます。すべてのリモートブランチおよびリモートのがされます。

リモートブランチ`dev`を`dev1`するには

```
git remote rename dev dev1
```

リモートリポジトリをする

リモートの`<name>`します。すべてのリモートブランチとリモートのがされます。

リモートリポジトリ`dev`をするには

```
git remote rm dev
```

リモートリポジトリをする

されたすべてのリモートリポジトリをするには、 `git remote`します。

したリモートハンドルのいがされます。

```
$ git remote
premium
premiumPro
origin
```

よりなをするには、 - `--verbose`または`-v`フラグをできます。にはURLとリモート `push`または`pull` のタイプがまれます

```
$ git remote -v
premiumPro    https://github.com/user/CatClickerPro.git (fetch)
premiumPro    https://github.com/user/CatClickerPro.git (push)
premium    https://github.com/user/CatClicker.git (fetch)
premium    https://github.com/user/CatClicker.git (push)
origin    https://github.com/ud/starter.git (fetch)
origin    https://github.com/ud/starter.git (push)
```

Gitリポジトリのリモートを

リモートリポジトリがされているは、これをうことができます。リモートURLをするコマンドはのとおりです。

```
git remote set-url
```

2つのをるのリモート、とURL。

のリモートURLをしてください

```
git remote -v
origin    https://bitbucket.com/develop/myrepo.git (fetch)
origin    https://bitbucket.com/develop/myrepo.git (push)
```

リモートURLをする

```
git remote set-url origin https://localserver/develop/myrepo.git
```

リモートURLをもうしてください

```
git remote -v
origin    https://localserver/develop/myrepo.git (fetch)
origin    https://localserver/develop/myrepo.git (push)
```

リモートリポジトリのをする

`git remote show <remote repository alias>`によってリモートリポジトリにするをできます。

```
git remote show origin
```

```
remote origin
Fetch URL:  https://localserver/develop/myrepo.git
Push  URL:  https://localserver/develop/myrepo.git
HEAD branch: master
Remote branches:
  master      tracked
Local branches configured for 'git pull':
  master      merges with remote master
Local refs configured for 'git push':
  master      pushes to master      (up to date)
```

オンラインでGit Remoteをむ https://riptutorial.com/ja/git/topic/4071/git-remote

8: Git rerere

き

`rerere` されたのでは、gitにのをえておくようできます。これにより、gitがじにしたときににされます。

Examples

をにする

`rerere`をにするには、のコマンドをします。

```
$ git config --global rerere.enabled true
```

これは、のリポジトリだけでなく、グロ―バルにもできます。

オンラインでGit rerereをむ https://riptutorial.com/ja/git/topic/9156/git-rerere

9: git send-email

- git send-email [options] <ファイル|ディレクトリ| rev-listオプション> ...
- git send-email - ダンプエイリアス

https://git-scm.com/docs/git-send-email

Examples

Gmailでgit send-emailをする

Linuxカーネルのようなプロジェクトでする、pullをすのではなく、コミットをlistservにしてレビューするがあります。このエントリーでは、Gmailでgit-sendメールをするについてしくします。

.gitconfigファイルにのをします。

```
[sendemail]
    smtpserver = smtp.googlemail.com
    smtpencryption = tls
    smtpserverport = 587
    smtpuser = name@gmail.com
```

に、ウェブでGoogle - >マイアカウント - >されたアプリとサイト - >のいアプリをする - >スイッチをオンにする

パッチセットをするには

```
git format-patch HEAD~~~~ --subject-prefix="PATCH <project-name>"
```

に、パッチをlistservにします。

```
git send-email --annotate --to project-developers-list@listserve.example.com 00*.patch
```

パッチのバージョンこのではバージョン2をしてするには

```
git format-patch -v 2 HEAD~~~~  ......
git send-email --to project-developers-list@listserve.example.com v2-00*.patch
```

- [no-] ccc - [no-] bcc *メールBcc--subject * Email "Subject" - -in-reply-to *メール "In-Reply-To" - [no-] xmailer * "X-Mailer"ヘッダーデフォルトをします。 - [no-] annotate *エディタでされるパッチをします。 --compose *のためにエディタをきます。 --compose-encoding *をしたエンコーディング。 -8bit-encoding *されていない、8bitメールをうためのエンコーディング--transfer-encoding *するエンコーディングquoted-printable、8bit、base64

パッチをメールでする

プロジェクトここではulogd2、ブランチはgit-svnにしてたくさんのコミットをしているとし、あなたのパッチセットをMaillingリストdevel@netfilter.orgにることをやめましょう。これをうには、gitディレクトリのルートでシェルをき、のをします。

```
git format-patch --stat -p --raw --signoff  --subject-prefix="ULOGD PATCH" -o /tmp/ulogd2/ -n
git-svn
git send-email --compose --no-chain-reply-to --to devel@netfilter.org /tmp/ulogd2/
```

のコマンドは、/ tmp / ulogd2 /のパッチからレポートをってのメールをし、2はエディタをしてパッチセットへのメールをします。ひどいスレッドされたメールシリーズをけるために、をできます。

```
git config sendemail.chainreplyto false
```

ソース

オンラインでgit send-emailをむ https://riptutorial.com/ja/git/topic/4821/git-send-email

10: Gitkをしてコミットをグラフィカルに

Examples

1つのファイルのコミットをする

```
gitk path/to/myfile
```

2つのコミットのすべてのコミットをする

`d9e1db9`と`5651067` 2つのコミットがあり、それらのでがこったのかをたいとしましょう。 `d9e1db9`はもいであり、 `5651067`はコミットチエーンのなです。

```
gitk --ancestry-path d9e1db9 5651067
```

バージョンタグにコミットをする

バージョンタグ`v2.3`をしているは、そのタグのすべてのコミットをできます。

```
gitk v2.3..
```

オンラインでGitkをしてコミットをグラフィカルにをむ https://riptutorial.com/ja/git/topic/3637/gitkをしてコミットをグラフィカルに

11: git-svn

にきな**SVN**リポジトリのクローン

git-svnがSVNリポジトリのなをするがあるため、SVNリポジトリがにきければ、このにかかることがあります。いにもSVNリポジトリをクローンするだけでみます。のgitリポジトリとに、repoフォルダをのにコピーすることもできます。のコンピュータにフォルダをコピーすると、きなSVNリポジトリをからクローンするだけですばやくできます。

コミットと**SHA1**について

`git svn dcommit`コマンドをすると、ローカルのgitコミットがきえられます。このコマンドは、SVNサーバーでされたSVNリビジョンをするgit commitメッセージにテキストをします。これはにです。しかし、しいテキストをするには、にはできないのコミットのメッセージをするがあります。gitコミットはできません。は、じとしいメッセージでしいコミットをしますが、にはしいコミットですつまり、git commitのSHA1がされます

git-svnのためにされたgitコミットはローカルなので、gitコミットのSHA1 idsはgitリポジトリごとになりますつまり、SHA1をってのからのコミットをすることはできません。じコミットでは、それぞれのローカルgitリポジトリになるSHA1があるからです。リポジトリのなるコピーでコミットをするは、SVNサーバーにプッシュするときにコミットメッセージにされたsvnリビジョンにするがあります。

SHA1をローカルにすることもできますのコミット、チェリーピックおよびリセットなどの/

トラブルシューティング

git svn rebaseコマンドがチェックサムのエラーをする

コマンドgit svn rebaseは、のようなエラーをします。

```
Checksum mismatch: <path_to_file> <some_kind_of_sha1>
expected: <checksum_number_1>
  got: <checksum_number_2>
```

こののは、のファイルがにされたときにsvnをリビジョンにリセットし、git svnをフェッチしてSVNをすることです。 SVNリセットをするコマンドはのとおりです。

- git log -1 - `<path_to_file>` コミットメッセージにされるSVNリビジョンをコピーします
- git svn reset `<revision_number>`
- git svn fetch

あなたはびSVNからデータをプッシュ/プルできるはずです

ファイルがコミットにつかりませんでしたSVNからフェッチまたはプルしようとすると、のようなエラーがされます

```
<file_path> was not found in commit <hash>
```

これは、SVNのリビジョンがらかのでローカルコピーにしないファイルをしようとしていることをします。このエラーをりくのは、ファイルのパスをしてフェッチをし、のSVNリビジョンのステータスにします

- `git svn fetch --ignore-paths <file_path>`

Examples

SVNリポジトリのクローニング

このコマンドでリポジトリのしいローカルコピーをするがあります

```
git svn clone SVN_REPO_ROOT_URL [DEST_FOLDER_PATH] -T TRUNK_REPO_PATH -t TAGS_REPO_PATH -b
BRANCHES_REPO_PATH
```

あなたのSVNリポジトリがレイアウトトランク、ブランチ、タグフォルダにっているなら、いくつかのタイプをすることができます

```
git svn clone -s SVN_REPO_ROOT_URL [DEST_FOLDER_PATH]
```

`git svn clone`は、SVNリビジョンを1つずつチェックアウトし、をするためにローカルリポジトリでgit commitをいます。 SVNリポジトリにくのコミットがある、これにはしばらくがかかります。

コマンドがすると、SVNリポジトリのトランクブランチをするmasterというローカルブランチをつなgitリポジトリがされます。

SVNからのをする

`git pull`するのはコマンドです

```
git svn rebase
```

これにより、SVNリポジトリからのすべてのがされ、のブランチのローカルコミットのにされます。

このコマンドをすることもできます

```
git svn fetch
```

SVNリポジトリからをしてローカルマシンにちみますが、ローカルブランチにはしません。

ローカルを**SVN**にプッシュする

コマンド

```
git svn dcommit
```

あなたのローカルgitコミットごとにSVNリビジョンをします。 SVNとに、ローカルのgitのはSVNリポジトリののとしているがあります。コマンドがしたは、に`git svn rebase`してみてください。

ローカルでく

のgitコマンドで、ローカルのgitリポジトリをのgitリポジトリとしてしてください

- `git add FILE`と`git checkout -- FILE`をステージング/アンステージする
- `git commit`をします。これらのコミットはローカルになり、のgitリポジトリとにSVNリポジトリに「プッシュ」されません
- `git stash`と`git stash pop`スタッシュをできるようにする
- `git reset HEAD --hard`あなたのすべてのローカルなをにす
- `git log`リポジトリのすべてのにアクセスする
- `git rebase -i`あなたののをにきえることができます
- `git branch`と`git checkout`をってローカルブランチをする

git-svnのドキュメントでは「SubversionはGitよりははるかにされていないシステムです」とべているので、Subversionサーバーのをすことなくgitのをすべてすることはできません。いにも、ルールはにです をにつ

これは、ほぼすべてのGitのをうことができますを、/べえ/コミットをし、コミットをし、りのをかす、などでもなく、マージ 。ローカルブランチのをするがあるは、わりに`git rebase`してください。

マージをすると、マージコミットがされます。マージコミットにするなことは、それらが2つのをち、それがをにすることです。は、あなたがリポジトリにマージコミットを「プッシュ」するにSVNをさせます。

しかし、しないでください **git merge commit**を**SVN**に「プッシュ」すると、もされません 。そうすると、git mergeコミットがsvnサーバーにされると、そのマージのすべてのコミットのすべてのがまれます。そのため、コミットのはわれますが、コードのはわれます。

のフォルダの

gitはフォルダのをせず、ファイルとそのファイルパスでします。つまり、gitはのフォルダをしません。しかし、SVNはそうです。 git-svnをうと、デフォルトで*git*でのフォルダをすると*SVN*にしません 。

コメントをするときに`--rmdir`フラグをすると、このがされ、SVNののファイルをローカルにする

とSVNののフォルダがされます。

```
git svn dcommit --rmdir
```

ながら、ののフォルダはされません。でうがあります。

dcommitをするたびにフラグをしないようにするには、git GUIツールSourceTreeなどをしているににするには、のコマンドをしてこのをデフォルトとしてします。

```
git config --global svn.rmdir true
```

これにより、.gitconfigファイルがされ、のがされます。

```
[svn]
rmdir = true
```

SVNのためにのままにしておくがある、すべてのuntrackedファイルとフォルダをするには、gitコマンドをいます

```
git clean -fd
```

してくださいのコマンドは、すべてののファイルとのフォルダ、SVNでするがあるものをしますあなたがSVNによってされたのフォルダをするがあるなら、コマンドをしてください

```
git svn mkdirs
```

これは、トラッキングされていないファイルやフォルダからワークスペースをクリーンアップするは、SVNがするのフォルダをするためににのコマンドをするがあることをします。

```
git clean -fd && git svn mkdirs
```

オンラインでgit-svnをむ https://riptutorial.com/ja/git/topic/2766/git-svn

12: git-tfs

Git-tfsは、GitリポジトリをTeam Foundation Server「TFS」リポジトリにするサードパーティのツールです。

ほとんどのリモートTFVSインスタンスは、すべてのでをし、Git-Credential-Manager-for-Windowsをインストールすることはにちません。あなたの`.git/config`あなたのとパスワードをすることでできます

```
[tfs-remote "default"]
  url = http://tfs.mycompany.co.uk:8080/tfs/DefaultCollection/
  repository = $/My.Project.Name/
  username = me.name
  password = My733TPwd
```

Examples

git-tfsクローン

これにより、プロジェクトとじのフォルダ、つまり/My.Project.Nameがされます。

```
$ git tfs clone http://tfs:8080/tfs/DefaultCollection/ $/My.Project.Name
```

git-tfsのクローン

gitリポジトリからのクローニングは、TFVSからのクローニングよりも10で、チームでうまくします。なくとも1のチームメンバーは、のgit-tfsクローンをにして、のgitリポジトリをするがあります。しいリポジトリは、TFVSでするようにブートストラップすることができます。

```
$ git clone x:/fileshare/git/My.Project.Name.git
$ cd My.Project.Name
$ git tfs bootstrap
$ git tfs pull
```

チョコレートをってgit-tfsをインストールする

は、ファイルdiffingにkdiff3をすることをとしていますが、ではありませんが、それはいえです。

```
C:\> choco install kdiff3
```

Gitはにインストールすることができますので、のパラメータをすることができます。ここでは、すべてのUnixツールもインストールされていて、 'NoAutoCrlf'はそのままのでコミットをします。

```
C:\> choco install git -params '"/GitAndUnixToolsOnPath /NoAutoCrlf"'
```

これはチョコレートでgit-tfsをインストールするためにになものです。

```
C:\> choco install git-tfs
```

git-tfsチェックイン

TFVSのチェックインダイアログをします。

```
$ git tfs checkintool
```

これにより、すべてのローカルコミットがされ、1つのチェックインがされます。

git-tfs push

すべてのローカルコミットをTFVSリモートにプッシュします。

```
$ git tfs rcheckin
```

チェックインノートがな、これはします。これは`git-tfs-force: rcheckin`をコミットメッセージにすることでできます。

オンラインでgit-tfsをむ https://riptutorial.com/ja/git/topic/2660/git-tfs

13: Gitクライアントサイドフック

き

のくのバージョンシステムとに、Gitはのなアクションがしたときにカスタムスクリプトをするをっています。これらのフックには、クライアントとサーバーの2つのグループがあります。クライアントのフックは、コミットやマージなどのによってトリガーされ、サーバーのフックはプッシュされたコミットのなどのネットワークでされます。これらのフックは、あらゆるでできます。

Examples

フックのりけ

フックはすべてGitディレクトリの`hooks`サブディレクトリにされています。ほとんどのプロジェクトでは、それは`.git/hooks`です。

フックスクリプトをにするには、`.git`ディレクトリの`hooks`サブディレクトリになしでなファイルをします。

Gitプレプッシュフック

プッシュプッシュスクリプトは、リモートステータスをチェックした、かが`git push`れるに、`git push`によってびされます。このスクリプトがゼロのですると、もプッシュされません。

このフックは、のパラメータでびされます。

```
 $1 -- Name of the remote to which the push is being done (Ex: origin)
 $2 -- URL to which the push is being done (Ex:
https://<host>:<port>/<username>/<project_name>.git)
```

プッシュされているコミットにするは、のでにとしてされます。

```
<local_ref> <local_sha1> <remote_ref> <remote_sha1>
```

サンプル

```
local_ref = refs/heads/master
local_sha1 = 68a07ee4f6af8271dc40caae6cc23f283122ed11
remote_ref = refs/heads/master
remote_sha1 = efd4d512f34b11e3cf5c12433bbedd4b1532716f
```

ののプッシュプッシュスクリプトは、デフォルトのpre-push.sampleからされました。これは、しいリポジトリが`git init`されたときににされました

```
# This sample shows how to prevent push of commits where the log message starts
# with "WIP" (work in progress).

remote="$1"
url="$2"

z40=0000000000000000000000000000000000000000

while read local_ref local_sha remote_ref remote_sha
do
    if [ "$local_sha" = $z40 ]
    then
        # Handle delete
        :
    else
        if [ "$remote_sha" = $z40 ]
        then
            # New branch, examine all commits
            range="$local_sha"
        else
            # Update to existing branch, examine new commits
            range="$remote_sha..$local_sha"
        fi

        # Check for WIP commit
        commit=`git rev-list -n 1 --grep '^WIP' "$range"`
        if [ -n "$commit" ]
        then
            echo >&2 "Found WIP commit in $local_ref, not pushing"
            exit 1
        fi
    fi
done

exit 0
```

オンラインでGitクライアントサイドフックをむ https://riptutorial.com/ja/git/topic/8654/gitクライアントサイドフック

14: Gitタグけ

き

ほとんどのバージョンシステムVCSとに、 `Git`はののポイントにな`tag`を`tag`があります。、このをしてリリースポイント `v1.0`などをマークします。

- gitタグ[-a | -s | -u <keyid>] [-f] [-m <msg> | -F <ファイル>] <タグ> [<コミット> | <object>]
- gitタグ-d <タグ>
- gitタグ[-n [<num>]] -l [--contains <commit>] [--contains <commit>] [--points-at <object>] [--column [= <オプション>] | - [no-] merged [<commit>]] [<pattern> ...] - [no-column] [--create-reflog] [--sort = <key>] [--format =]
- gitタグ-v [--format = <フォーマット>] <タグ> ...

Examples

なすべてのタグをする

コマンド`git tag`をすると、なすべての`git tag`されます。

```
$ git tag
<output follows>
v0.1
v1.3
```

`tags`はアルファベットにされます。

な`tags search`することもでき`tags`

```
$ git tag -l "v1.8.5*"
<output follows>
v1.8.5
v1.8.5-rc0
v1.8.5-rc1
v1.8.5-rc2
v1.8.5-rc3
v1.8.5.1
v1.8.5.2
v1.8.5.3
v1.8.5.4
v1.8.5.5
```

GITでタグをしてプッシュする

タグをする

- のブランチにタグをするには

```
git tag < tagname >
```

これにより、のブランチののをつローカル`tag`がされます。

- コミットをむタグをするには

```
git tag tag-name commit-identifier
```

これにより、のブランチのコミットをつローカル`tag`がされます。

GITでコミットをプッシュ

- 々のタグをプッシュ

```
git push origin tag-name
```

- にすべてのタグをプッシュする

```
git push origin --tags
```

オンラインでGitタグけをむ https://riptutorial.com/ja/git/topic/10098/gitタグけ

15: Gitののディレクトリ

Examples

Gitはディレクトリをしません

のディレクトリをつプロジェクトをしたとします。

```
/build
app.js
```

に、これまでにしたものをすべてしてコミットします。

```
git init
git add .
git commit -m "Initial commit"
```

Gitはapp.jsファイルのみをします

あなたがあなたのアプリケーションにビルドステップをしたとし、ディレクトリとしてそこに「ビルド」ディレクトリにってとあなたはすべてのがわなければならないこと、セットアップしたくない、 はめることですディレクトリに ".gitkeep"ファイルをき、Gitにそのファイルをさせます。

```
/build
  .gitkeep
app.js
```

に、このしいファイルをします。

```
git add build/.gitkeep
git commit -m "Keep the build directory around"
```

Gitはファイルビルド/ .gitkeepファイルをするので、ビルドフォルダはチェックアウトににになります。

りしになりますが、これはなるであり、Gitのではありません。

オンラインでGitののディレクトリをむ https://riptutorial.com/ja/git/topic/2680/gitののディレクトリ

16: Gitへの

Examples

AtlassianユーティリティをしてSVNからGitにする

ここで Atlassianユーティリティをダウンロードしてください 。このユーティリティにはJavaがです。をうマシンにJava Runtime Environment JREがインストールされていることをしてください。

`java -jar svn-migration-scripts.jar verify`コマンドをして、をするためになプログラムがマシンにないか`java -jar svn-migration-scripts.jar verify`をします。には、このコマンドはGit、Subversion、および`git-svn`ユーティリティをチェックします。また、をするファイルシステムでをしていることもします。 Gitへのは、リポジトリのをけるため、をするファイルシステムでうがあります。

に、ファイルをするがあります。 Subversionはコミッターのユーザーだけでをします。しかし、Gitはユーザをするために2つの、すなわちとメールアドレスをします。のコマンドは、SubversionユーザーをGitにするものにマッピングするテキストファイルをします。

```
java -jar svn-migration-scripts.jar authors <svn-repo> authors.txt
```

ここで`<svn-repo>`はしたいSubversionリポジトリのURLです。このコマンドをすると、のが`authors.txt`にマップされます。メールアドレスは`<username>@mycompany.com`になります。 authorsファイルでは、ユーザーのデフォルトデフォルトではユーザーになっていますをのにでするがあります。するに、すべてのメールアドレスのしさをしてください。

のコマンドは、svn repoをGitとしてクローンします

```
git svn clone --stdlayout --authors-file=authors.txt <svn-repo> <git-repo-name>
```

ここで、 `<svn-repo>`はでしたものとじリポジトリURLで、 `<git-repo-name>`はリポジトリをするカレントディレクトリのフォルダです。このコマンドをするに、いくつかのがあります。

- の`--stdlayout`フラグは、Gitに、 `trunk` 、 `branches` 、および`tags`フォルダをつレイアウトをしていることをえます。レイアウトのSubversionリポジトリでは、 `trunk`フォルダ、の/すべてのbranchフォルダ、および`tags`フォルダのをするがあります。 `git svn clone --trunk=/trunk --branches=/branches --branches=/bugfixes --tags=/tags --authors-file=authors.txt <svn-repo> <git-repo-name>` 。
- このコマンドは、レポのサイズにじてするまでにかかることがあります。
- リポジトリのをするために、はネットワークオーバーヘッドをなくすためにSubversionリポジトリをホストするサーバでできます。

`git svn clone`は、Subversionタグ`tags/`がにいたリモートブランチをむリモートブランチとしてSubversionブランチおよびトランクをインポートし`tags/`。これらをのブランチとタグにするには、Linuxマシンでのコマンドをそれらのにします。それらをした、`git branch -a`はしいブランチをし、`git tag -l`はリポジトリタグをします。

```
git for-each-ref refs/remotes/origin/tags | cut -d / -f 5- | grep -v @ | while read tagname;
do git tag $tagname origin/tags/$tagname; git branch -r -d origin/tags/$tagname; done
git for-each-ref refs/remotes | cut -d / -f 4- | grep -v @ | while read branchname; do git
branch "$branchname" "refs/remotes/origin/$branchname"; git branch -r -d "origin/$branchname";
done
```

svnからGitへのがしましたロカルリポジトリをサバに`push`するだけで、Gitをってしけることができ、svnからにされたバジョンをつこともできます。

サブギット

SubGitは、SVNリポジトリをgitにだけインポートするためにできます。

```
$ subgit import --non-interactive --svn-url http://svn.my.co/repos/myproject myproject.git
```

svn2gitをってSVNからGitにする

svn2gitは、SubversionからGitにプロジェクトをし、トランク、タグ、ブランチをむをするのにつ、git-svnによるgitのネイティブSVNサポートにするRubyラッパーです。

レイアウトつまり、リポジトリのルートレベルにあるブランチ、タグ、トランクでsvnリポジトリをするには

```
$ svn2git http://svn.example.com/path/to/repo
```

レイアウトにないsvnリポジトリをするには

```
$ svn2git http://svn.example.com/path/to/repo --trunk trunk-dir --tags tags-dir --branches
branches-dir
```

ブランチ、タグ、またはトランクをしないまたはっていないは、オプション`--notrunk`、-- `--nobranches`、および`--notags`できます。

たとえば、`$ svn2git http://svn.example.com/path/to/repo --trunk trunk-dir --notags --nobranches`はトランクのみをします。

しいリポジトリになスペースをらすために、したディレクトリやファイルをすることができますビルドディレクトリやアカイブなど。

```
$ svn2git http://svn.example.com/path/to/repo --exclude build --exclude '.*\.zip$'
```

の

しくしたgitリポジトリにすでにまたはそれのコミットがあるは、リモートでリポジトリをプッシュするに、するをらすことができます。これは、のコマンドをしてできます。

```
$ git gc --aggressive
```

のコマンドは、きなリポジトリコミット、および/またはMBのでかかるがあります。

Team Foundation Version ControlTFVCからGitにする

Git-TFというオープンソースツールをして、チームファンデーションのバージョンからgitにすることができます。では、tfsチェックインをgitコミットにすることで、のをします。

Git-TFをってソリューションをGitにれるには、のにいます。

Git-TFをダウンロード

Codeplex Git-TF @ CodeplexからGit-TFをダウンロードおよびインストールすることができます。

TFVCソリューションのクローン

powershellをしwin、コマンドをします。

```
git-tf clone http://my.tfs.server.address:port/tfs/mycollection
'$/myproject/mybranch/mysolution' --deep
```

--deepスイッチは、Git-Tfにあなたのチェックインをコピーするようするので、すべきkeeywordです。 cloeコマンドをびしたフォルダに、ローカルのgitリポジトリがあります。

- .gitignoreファイルをします。 Visual Studioをしているは、エディタがこれをうことができます。そうでなければ、 github / gitignoreからなファイルをダウンロードすることででうことができます。
- ソリューションからTFSソースバインディングをしますすべての* .vssccファイルをします。 GlobalSectionTeamFoundationVersionControlをして、ソリューションファイルをすることもできます。..... EndClobalSection

コミットプッシュ

ローカルリポジトリをコミットしてリモートにプッシュすることで、をします。

```
git add .
git commit -a -m "Coverted solution source control from TFVC to Git"

git remote add origin https://my.remote/project/repo.git

git push origin master
```

MercurialからGitへの

`Mercurial` Repoを`Git`にインポートするには、のメソッドをできます

1. エクスポートの

```
cd
git clone git://repo.or.cz/fast-export.git
git init git_repo
cd git_repo
~/fast-export/hg-fast-export.sh -r /path/to/old/mercurial_repo
git checkout HEAD
```

2. Hg-Gitのになえはhttps://stackoverflow.com/a/31827990/5283213です。

3. いのGitHubのを でのにってくださいGitHubの 。

オンラインでGitへのをむ https://riptutorial.com/ja/git/topic/3026/gitへの

17: Gitラージファイルストレージ LFS

Git Large File Storage LFSは、なファイル、にバイナリをバージョンするのパフォーマンスがいというGitバージョンシステムのをすることをとしています。 LFSはこのようなファイルのをサーバにし、わりにgitオブジェクトデータベースのそれらのアセットのパスへのテキストポインタをコミットするだけで、このをします。

LFSをしてされるなファイルタイプは、コンパイルされたソースとなるがあります。 PSDやJPEGなどのグラフィックアセット、または3D。このようにして、プロジェクトによってされるリソースは、のシステムをにするのではなく、じリポジトリでることができます。

LFSはもともとGitHub https://github.com/blog/1986-announcing-git-large-file-storage-lfsによってされました。しかし、Atlasssianはほぼじにgit-lobとばれるのプロジェクトにりんでいました。もなく、これらのは、のをけるためにされました。

Examples

LFSをインストールする

HomebrewまたはWebサイトからダウンロードしてインストールします 。

BREWの、

```
brew install git-lfs
git lfs install
```

くの、リモートをホストするサービスでlfsでするようにするもあります。これはホストごとになりますが、git lfsをしたいというボックスをチェックしているがあります。

にするのファイルタイプをする

Git LFSをするなワークフローは、 `.gitignore`ファイルとに、ルールベースのシステムをしてされるファイルをすることです。

くの、ワイルドカードはブランケットトラックにのファイルタイプをふためにされます。

えば`git lfs track "*.psd"`

のパターンとするファイルがコミットされたときに、それがリモートにプッシュされると、リモートリポジトリのファイルをきえるポインタで々にアップロードされます。

`.gitattributes`ファイルをすると、 `.gitattributes`ファイルがそれにじてされます。グローバルな`.gitattributes`ファイルでするのではなく、ローカルの`.gitattributes`ファイルをコミットすることをおめします。これにより、なるプロジェクトでするときにがしないようになります。

すべてのクローンのLFSをする

すべてのクローンにされるLFSオプションをするには、リポジトリルートに`.lfsconfig`というのファイルをしてコミットします。このファイルは`.git/config`されるのとじでLFSオプションをできます。

たとえば、LFSフェッチからのファイルをするには、デフォルトで、 `.lfsconfig`をしてのでコミットします。

```
[lfs]
    fetchexclude = ReallyBigFile.wav
```

オンラインでGitラージファイルストレージLFSをむ https://riptutorial.com/ja/git/topic/4136/gitラージファイルストレージ-lfs-

18: Gitリビジョンの

くのGitコマンドは、リビジョンパラメータをとしてります。コマンドによっては、のコミット、またはリビジョングラフをくコマンド git-log1など のは、そのコミットからなすべてのコミットをします。これらは、ので`<commit>` 、 `<rev>` 、または`<revision>`れます。

Gitリビジョンのリファレンスドキュメントは、 gitrevisions7のマンページです。

まだこのページからけている

- [ ] `git describe`からの `v1.7.4.2-679-g3bee7fb`
- [ ] `@`だけで`HEAD`ショートカット
- [ ] `@{-<n>}` 、えば`@{-1}` 、および`-@{-1}`
- [ ] `<branchname>@{push}`
- [ ] `<rev>^@` 、すべてののための`<rev>`

のドキュメントがです。

- [ ]リポジトリおよびインデックスのブロブおよびツリーをする `<rev>:<path>`および `:<n>:<path>`
- [ ]リビジョンはのように`A..B` 、 `A...B` 、 `B ^A` 、 `A^1` 、び`-<n> --since`

Examples

オブジェクトによるリビジョンの

```
$ git show dae86e1950b1277e545cee180551750029cfe735
$ git show dae86e19
```

SHA-1オブジェクト、な40バイトの16、またはリポジトリにのをしてリビジョンまたはにはのオブジェクトタグ、ツリーすなわちディレクトリの、ブロブ、すなわちファイルのをできます。

シンボルブランチ、タグ、リモートトラッキングブランチ

```
$ git log master    # specify branch
$ git show v1.0     # specify tag
$ git show HEAD     # specify current branch
$ git show origin   # specify default remote-tracking branch for remote 'origin'
```

ブランチえば 'master'、 'next'、 'maint'、タグえば 'v1.0'、 'v0.6.3-rc2'、remote- 'origin'、 'origin / master'などのブランチ、およびのブランチの 'HEAD'などのなをします。

シンボリックリファレンスがあいまいである、たとえばブランチとタグのが 'fix'ブランチとタグののものはされていないのは、するリファレンスのをするがあります。

```
$ git show heads/fix      # or 'refs/heads/fix', to specify branch
$ git show tags/fix       # or 'refs/tags/fix', to specify tag
```

デフォルトのリビジョンHEAD

```
$ git show         # equivalent to 'git show HEAD'
```

'HEAD'は、ツリーのにづいたコミットにをけます。、のブランチのシンボリックです。リビジョンパラメータをるくのコマンドすべてではないは、デフォルトで 'HEAD'になっています。

Reflog @ { }

```
$ git show @{1}            # uses reflog for current branch
$ git show master@{1}      # uses reflog for branch 'master'
$ git show HEAD@{1}        # uses 'HEAD' reflog
```

ref、はブランチまたはHEADのに`@`つけ、のペアでまれたのえば`{1}` 、 `{15}` はローカルリポジトリのそのrefのnののをします 。のreflogエントリは`git reflog`コマンドでチェックするか、 `--walk-reflogs` / `-g`オプションを`--walk-reflogs`て`git log`チェックすることができ`git log`。

```
$ git reflog
08bb350 HEAD@{0}: reset: moving to HEAD^
4ebf58d HEAD@{1}: commit: gitweb(1): Document query parameters
08bb350 HEAD@{2}: pull: Fast-forward
f34be46 HEAD@{3}: checkout: moving from af40944bda352190f05d22b7cb8fe88beb17f3a7 to master
af40944 HEAD@{4}: checkout: moving from master to v2.6.3

$ git reflog gitweb-docs
4ebf58d gitweb-docs@{0}: branch: Created from master
```

reflogをすると、 `ORIG_HEAD` ref `HEAD@{1}`ほぼをするいメカニズムがにきえられました。

Reflog @ { }

```
$ git show master@{yesterday}
$ git show HEAD@{5 minutes ago}   # or HEAD@{5.minutes.ago}
```

refにいて、のをのペアでんだ`@`がきます `{yesterday}` 、 `{1 month 2 weeks 3 days 1 hour 1 second ago}` `{1979-02-26 18:30:00}` `{1 month 2 weeks 3 days 1 hour 1 second ago}`または`{1979-02-26 18:30:00}` のまたはそれにもいでのrefの。これは、されたにあなたのローカル refのをべることにしてください。あなたのの「マスター」にいたことがあります。

`git reflog`を`git reflog`とともにすると、ローカルリポジトリでしたになをすることができます。

```
$ git reflog HEAD@{now}
08bb350 HEAD@{Sat Jul 23 19:48:13 2016 +0200}: reset: moving to HEAD^
4ebf58d HEAD@{Sat Jul 23 19:39:20 2016 +0200}: commit: gitweb(1): Document query parameters
08bb350 HEAD@{Sat Jul 23 19:26:43 2016 +0200}: pull: Fast-forward
```

されたのブランチ @{の}

```
$ git log @{upstream}..       # what was done locally and not yet published, current branch
$ git show master@{upstream}  # show upstream of branch 'master'
```

にけられた`@{upstream}`は、 `<branchname>@{u}` でされたがにされるようにされているをします `branch.<name>.remote`と`branch.<name>.merge` 、または`git branch --set-upstream-to=<branch>` 。 しているはのデフォルトになります。

リビジョンのとに、あなたのブランチがにんじているコミットのリポジトリにコミットしていないコミットと、あなたがコミットしているもののコミットがロ―カルブランチにマ―ジされていない

```
$ git log --oneline @{u}..
$ git log --oneline ..@{u}
$ git log --oneline --left-right @{u}...  # same as ...@{u}
```

をコミットする ^、

```
$ git reset --hard HEAD^             # discard last commit
$ git rebase --interactive HEAD~5    # rebase last 4 commits
```

`^`は、コミット･オブジェクトののをします。 `^<n>`は<n>のをしますすなわち、 `<rev>^`は`<rev>^1`とです。

`~<n>`をリビジョン･パラメ―タにすると、のにく、されたコミット･オブジェクトの<n>であるコミット･オブジェクトをします。つまり、 `<rev>~3`は`<rev>^^^`とです。ショ―トカットとして、 `<rev>~<rev>~1` 、およびです`<rev>^1` 、または`<rev>^`いインチ

このはです。

そのようなをつけるには、 `git name-rev`コマンドをいます

```
$ git name-rev 33db5f4d9027a10e477ccf054b2c1ab94f74c85a
33db5f4d9027a10e477ccf054b2c1ab94f74c85a tags/v0.99~940
```

その`--pretty=oneline`していない`--oneline` 、のでするがあります

```
$ git log --pretty=oneline | git name-rev --stdin --name-only
master Sixth batch of topics for 2.10
master~1 Merge branch 'ls/p4-tmp-refs'
master~2 Merge branch 'js/am-call-theirs-theirs-in-fallback-3way'
[...]
master~14^2 sideband.c: small optimization of strbuf usage
master~16^2 connect: read $GIT_SSH_COMMAND from config file
[...]
master~22^2~1 t7810-grep.sh: fix a whitespace inconsistency
master~22^2~2 t7810-grep.sh: fix duplicated test name
```

とタグをする ^0、^{}

コマンドのは、ブランチ、タグ、またはのリビジョンがされているかどうかによってなります。がなは、「」をできます。

サフィックス^オブジェクトタイプく`tag`、`commit`、`tree`、`blob`ブレースペアでまがえば、`v0.99.8^{commit}`でオブジェクトををする`<rev>`のオブジェクトまで`<type>`オブジェクトがつからないか、オブジェクトがされなくなりました。`<rev>^0`は`<rev>^{commit}`です。

```
$ git checkout HEAD^0    # equivalent to 'git checkout --detach' in modern Git
```

^ろにの`v0.99.8^{}`たとえば、`v0.99.8^{}`は、タグオブジェクトがつかるまで、タグをにすることをします。

```
$ git show v1.0
$ git cat-file -p v1.0
$ git replace --edit v1.0
```

と

```
$ git show v1.0^{}
$ git cat-file -p v1.0^{}
$ git replace --edit v1.0^{}
```

いマッチングコミット ^{/}、/

```
$ git show HEAD^{/fix nasty bug}   # find starting from HEAD
$ git show ':/fix nasty bug'       # find starting from any branch
```

コロン':'のにスラッシュ'/'をけ、そのにテキストをけると、コミットメッセージがされたにするコミットにがけられます。このは、のマッチングのからであるコミットをします。は、コミットメッセージのどのにもします。でまるメッセージをするには、えば`:/^foo`します。なシーケンス`:/!`するもののとしてされています。`:/!-foo`はのマッチをいますが、`:/!!foo`はリテラルにマッチします、いて`foo`。

^はリビジョンパラメータにき、スラッシュでまれたテキストをむのには、`<rev>`からなもいマッチをすの`:/<text>`とじです。^。

オンラインでGitリビジョンのをむ https://riptutorial.com/ja/git/topic/3735/gitリビジョンの

19: gitリポジトリをする

き

githubやbitbucketのようなリモートのリポジトリをすると、のコードをプッシュするとエラーなエラー、リポジトリがつかりません。**がされます。

Examples

ローカルをする

ターミナルにく、

```
cd projectFolder
git remote -v (it will show previous git url)
git remote set-url origin https://username@bitbucket.org/username/newName.git
git remote -v (double check, it will show new git url)
git push (do whatever you want.)
```

オンラインでgitリポジトリをするをむ https://riptutorial.com/ja/git/topic/9291/gitリポジトリをする

20: Git

- git log [<options>] [<revision range>] [[-] <path>]
- git log --pretty = short | git shortlog [<options>]
- git shortlog [<options>] [<revision range>] [[-] <path>]

パラ メーター

パラ メーター	
-n 、 -- --numbered	アルファベットではなく、ごとのコミットにってをソートする
-s 、 - --summary	コミット·カウントののみをする
-e 、 -- --email	のメールアドレスをする
--format [= <>]	コミットのわりに、のをして、コミットをします。 <format>には 、 git log --formatオプションでけれられるのを--formatできgit log 。
-w [<width> [、 <indent1> [、 <indent2>]]]	をwidthりしてをりします。エントリののは、によってインデントされindent1スペースの、およびそれにくがでインデントさindent2スペース。
<リビジョン>	したリビジョンのコミットのみをします。のコミットまでののデフォルトです。
[--] <パス>	pathするファイルがどのようになったかをするコミットのみをします。オプションやリビジョンからパスをるには、として " - "をけるがあります。

Examples

ごとのコミット

Gitのshortlogログは、gitのログをし、コミットをshortlogにまとめるためにされます。

デフォルトでは、すべてのコミットメッセージがされますが、--summaryまたは-sはメッセージをスキップし、コミットをつのリストをします。

--numberedまたは-nは、アルファベットからのコミットにをします。

```
git shortlog -sn        #Names and Number of commits
```

```
git shortlog -sne       #Names along with their email ids and the Number of commits
```

または

```
git log --pretty=format:%ae \
| gawk -- '{ ++c[$0]; } END { for(cc in c) printf "%5d %s\n",c[cc],cc; }'
```

じによるコミットは、やメールアドレスのスペルがなるところでグループすることはできません。たとえば、 `John Doe`と`Johnny Doe`がリストに々にされます。これをするには、 `.mailmap`をしてください。

ごとにコミットする

```
git log --pretty=format:"%ai" | awk '{print " : "$1}' | sort -r | uniq -c
```

ブランチのコミットの

```
git log  --pretty=oneline |wc -l
```

ブランチとそののリビジョンのをリストする

```
for k in `git branch -a | sed s/^..//`; do echo -e `git log -1 --pretty=format:"%Cgreen%ci
%Cblue%cr%Creset" $k --`\\t"$k";done | sort
```

ごとのコード

```
git ls-tree -r HEAD | sed -Ee 's/^.{53}//' | \
while read filename; do file "$filename"; done | \
grep -E ': .*text' | sed -E -e 's/: .*//' | \
while read filename; do git blame --line-porcelain "$filename"; done | \
sed -n 's/^author //p' | \
sort | uniq -c | sort -rn
```

すべてのコミットをかなりのでする

```
git log --pretty=format:"%Cgreen%ci %Cblue%cn  %Cgreen%cr%Creset %s"
```

これにより、、ユーザー、コミットメッセージをむすべてのコミット1に1つのらしいがられます。

`--pretty`オプションにはくのプレースホルダがあり、それぞれ%まります。すべてのオプションはここにあります

コンピュータのすべてのローカルGitリポジトリをする

gitリポジトリのすべてのをするには、のコマンドをします

```
find $HOME -type d -name ".git"
```

`locate`ているとすると、これははるかにくなるはずです。

```
locate .git |grep git$
```

あなたがっているは`gnu locate`や`mlocate`、これがのgitのdirsをします

```
locate -ber \\.git$
```

ごとのコミットのをする

それぞれのやがリポジトリにったコミットのをするには、に`git shortlog`します。

```
git shortlog -s
```

とそれぞれのコミットをします。

さらに、がすべてのブランチでされるようにするには、コマンドに`--all`フラグをします。

```
git shortlog -s --all
```

オンラインでGitをむ https://riptutorial.com/ja/git/topic/4609/git

21: GUIクライアント

Examples

GitHubデスクトップ

ウェブサイト https : //desktop.github.com

プラットフォームOS XおよびWindows
GitHub

ギットクラッケン

ウェブサイト https : //www.gitkraken.com
$ 60 /オープンソース、、、、のは
プラットフォームLinux、OS X、Windows
Axosoft

SourceTree

ウェブサイト https : //www.sourcetreeapp.com
アカウントが
プラットフォームOS XおよびWindows
Atlassian

gitkとgit-gui

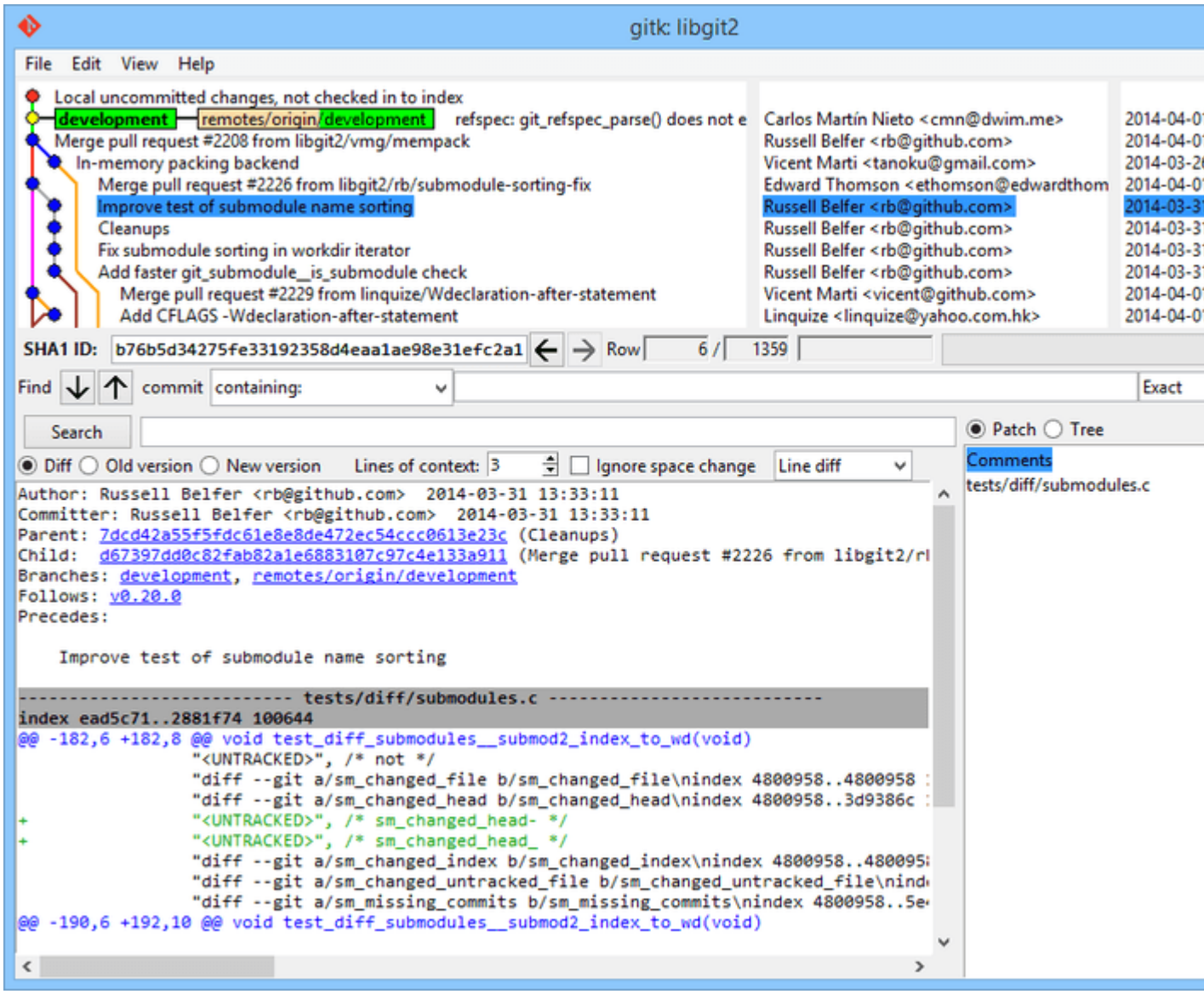
Gitをインストールすると、そのツールであるgitkとgit-guiもできます。

`gitk`はグラフィカルなビューアです。それはgit logとgit grepのなGUIシェルのようにえてください。これは、にこったことをつけようとしているときや、プロジェクトのをするときにするツールです。

Gitkはコマンドラインからびすのがもです。 Gitリポジトリにcdしてのようにしてください

```
$ gitk [git log options]
```

Gitkは、くのコマンドラインオプションをけれます。ほとんどのコマンドラインオプションは、になるgitログアクションにされます。おそらくもなのは、 `--all`フラグです。これはgitkにHEADだけでなく、どのからもなコミットをするようにします。 Gitkのインターフェースはのようになります



1-1。 gitkビューア。

にgit log --graphののようなものがあります。ドットはコミットをし、はをし、refはきのボックスとしてされます。のはHEADをし、のはまだコミットされていないをします。には、されたコミットのビュ—があります。コメントとパッチはに、サマリ—ビュ—はにされます。には、のにされるコントロ—ルのコレクションがあります。

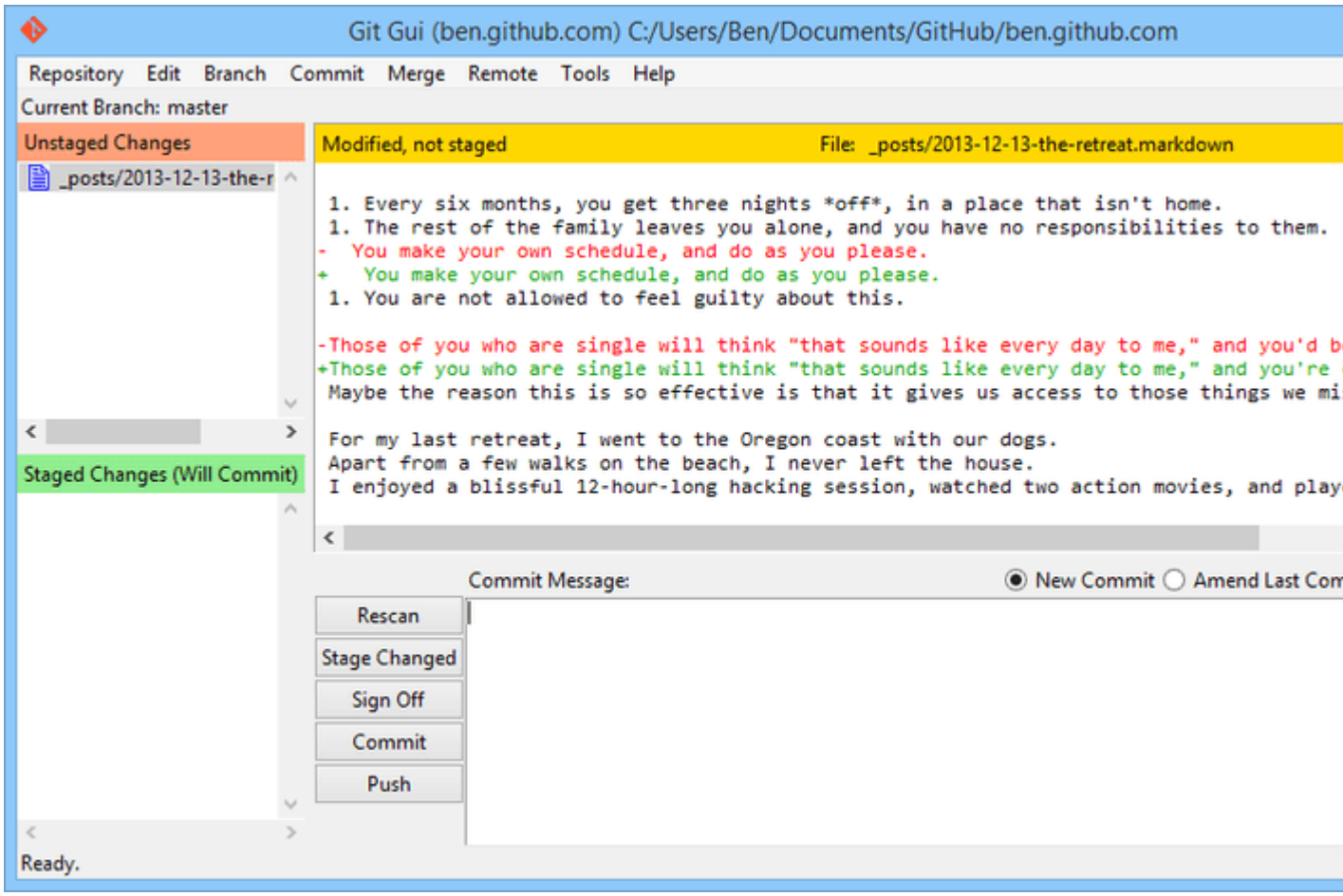
ブランチやコミットメッセ—ジをクリックすると、gitのくのにアクセスできます。たとえば、のブランチやチェリ—をチェックアウトすることは、ワンクリックでにできます。

、 `git-gui` 、にコミットをするためのツ—ルです。これもコマンドラインからびすのがもです

```
$ git gui
```

そして、これはのようになります

`git-gui`コミットツ—ル。



1-2。 git-guiコミットツール。

にインデックスがあります。ステージにないがにされ、にステージングされたがあります。 2つののでファイルをするには、アイコンをクリックするか、をクリックしてするファイルをします。

には、されているファイルのをすビューがあります。このエリアをクリックすることで、々のハンクまたは々のラインをステージングすることができます。

にメッセージとアクションのがあります。テキストボックスにメッセージをして、"コミット"をクリックしてgit commitとのことをします。のコミットのを「」にする「」ラジオボタンをして、のコミットをすることもできます。に、をステージングまたはステージングし、コミットメッセージをして、もう「コミット」をクリックして、いコミットをしいものにきえます。

gitkとgit-guiはタスクのツールのです。それぞれはののとコミットのにわせてカスタマイズされ、そのタスクにはなはされています。

ソース https：//git-scm.com/book/en/v2/Git-in-Other-Environments-Graphical-Interfaces

SmartGit

ウェブサイト http：//www.syntevo.com/smartgit/
でのみです。ライセンスは99ドルです

プラットフォームLinux、OS X、Windows
syntevo

Git

ウェブサイト https : //gitextensions.github.io

プラットフォームWindows

オンラインでGUIクライアントをむ https://riptutorial.com/ja/git/topic/5148/guiクライアント

22: Reflog - git logにされていないコミットをする

Gitのreflogは、されるたびにHEADリポジトリのののののをします。に、であるのあるすべてのにはHEADポインタのがまれますのものをめ、らかのがあった、チップコミットのハッシュがされるため、ながわれるににのにすことができます reflogののしいをつけてください。

しかし、refでされていないオブジェクトは30にガベ―ジコレクションされるため、reflogがいつもけになるとはりません。

Examples

いrebaseからのリカバリ

インタラクティブなリベ―スをしたとします。

```
git rebase --interactive HEAD~20
```

って、あなたはいたくないコミットをしつぶしたりとしたりしましたが、リベ―スをしました。するには、 `git reflog` 、のようながされることがあります。

```
aaaaaaa HEAD@{0} rebase -i (finish): returning to refs/head/master
bbbbbbb HEAD@{1} rebase -i (squash): Fix parse error
...
ccccccc HEAD@{n} rebase -i (start): checkout HEAD~20
ddddddd HEAD@{n+1} ...
...
```

この、のコミット`ddddddd` または`HEAD@{n+1}` は、あなたのプリ*rebase*ブランチのです。したがって、そのコミットおよびすべてのコミットにしつぶされたりとされたものをむをするには、のようにします。

```
$ git checkout HEAD@{n+1}
```

その、 `git checkout -b [branch]`をってそのコミットでしいブランチをすることができます。は、「 」をしてください。

オンラインでReflog - git logにされていないコミットをするをむ
https://riptutorial.com/ja/git/topic/5149/reflog-----git-logにされていないコミットをする

23: Revリスト

- git rev-list [オプション] <コミット> ...

パラメーター

パラメータ	
- オンライン	は、タイトルとともに1にコミットされます。

Examples

マスタのコミットをリストしますが、/マスタはコミットしません

```
git rev-list --oneline master ^origin/master
```

Git rev-listは、のブランチにないブランチのコミットをリストします。これは、コードがブランチにマージされているかどうかをべようとしているときにはらしいツールです。

- `--oneline`オプションをすると、コミットのタイトルがされます。
- `^`は、されたブランチのコミットをリストからします。
- 2つのブランチをすことができます。えば、 `git rev-list foo bar ^baz`は、fooとbarにコミットしますが、 `git rev-list foo bar ^baz`はコミットしません。

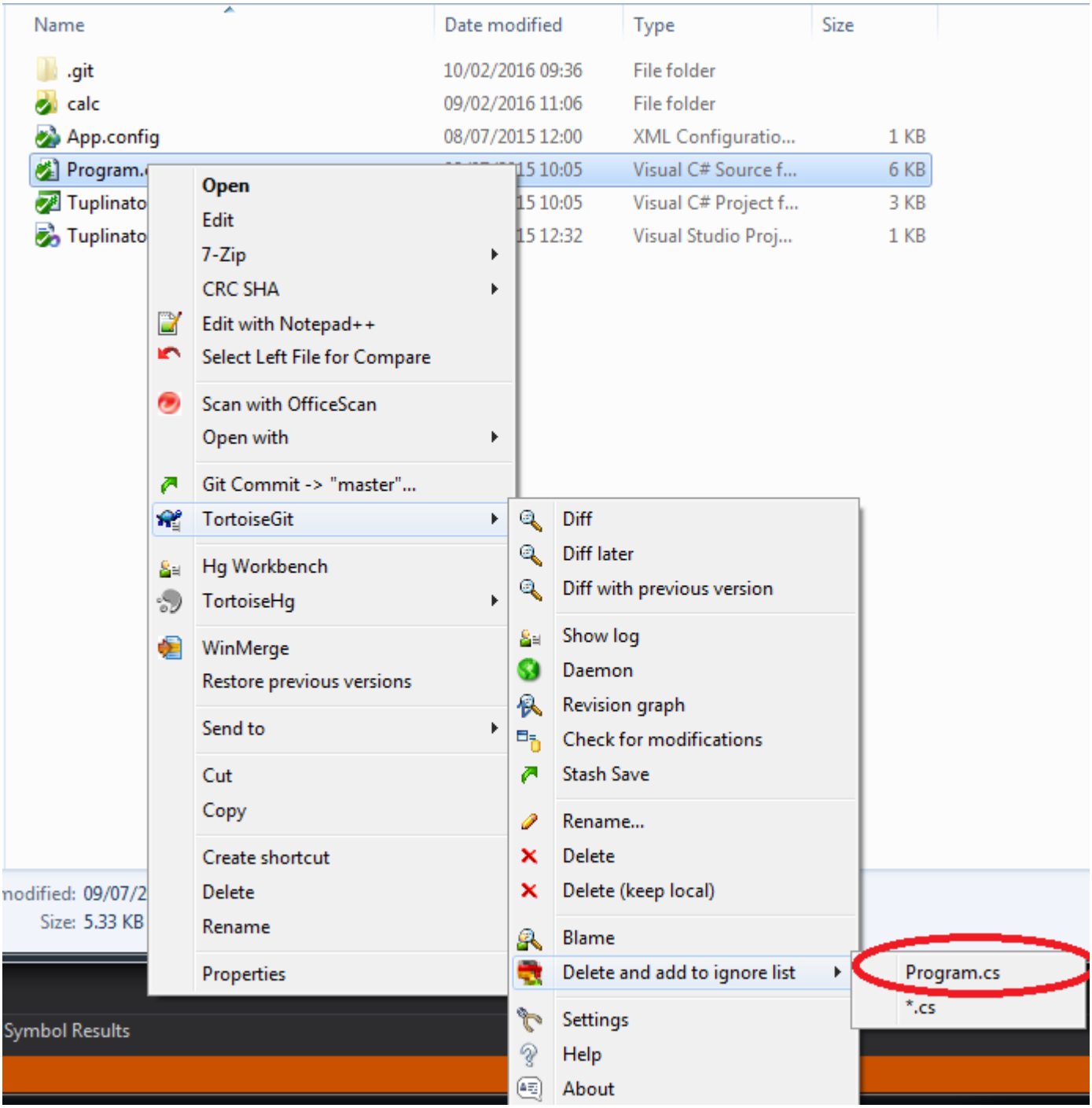
オンラインでRevリストをむ https://riptutorial.com/ja/git/topic/431/revリスト

24: TortoiseGit

Examples

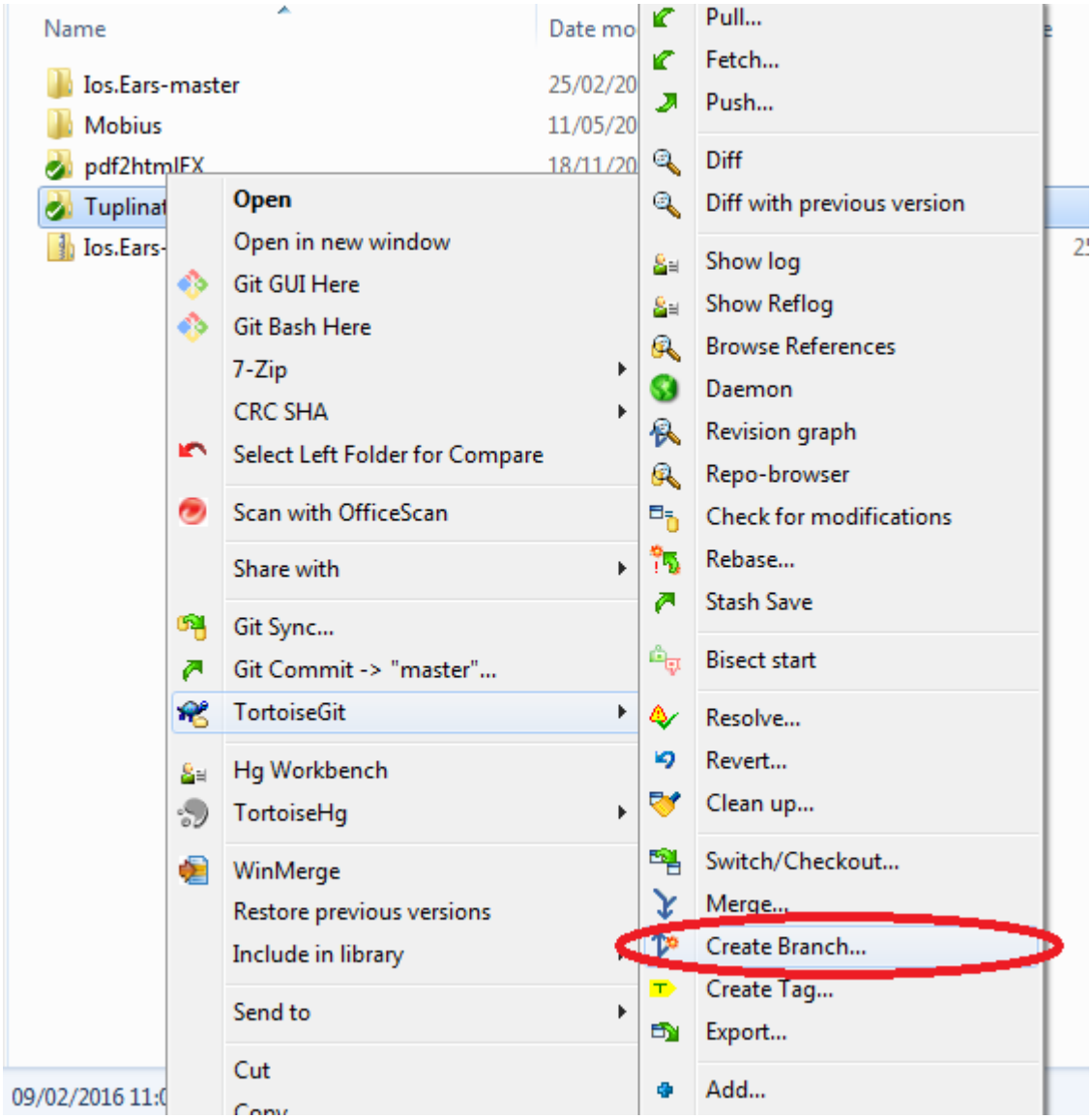
ファイルとフォルダをする

TortioseGit UIをしているは、するファイルまたはフォルダを□クリックします -> `TortoiseGit` -> `Delete and add to ignore list`に`Delete and add to ignore list`、ここでそののファイルまたはこのファイルをすることができます ->ダイアログ`Ok`をクリックするとします。

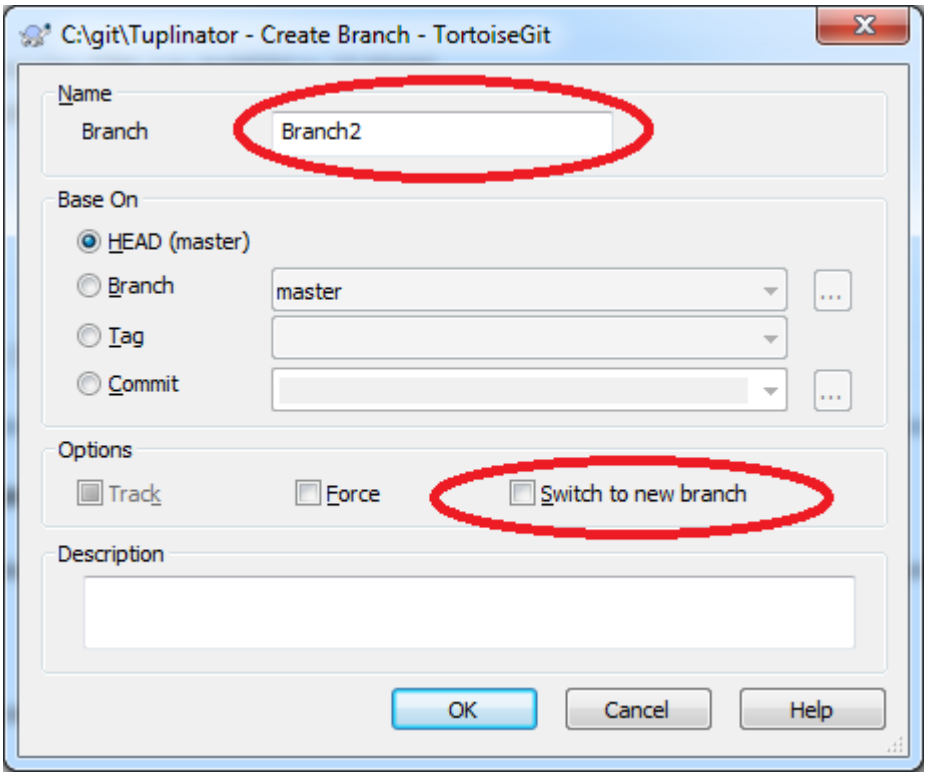


リポジトリをクリックして□マウスをクリックすると、 `Tortoise Git` -> `Create Branch...` UIをして

いるのために

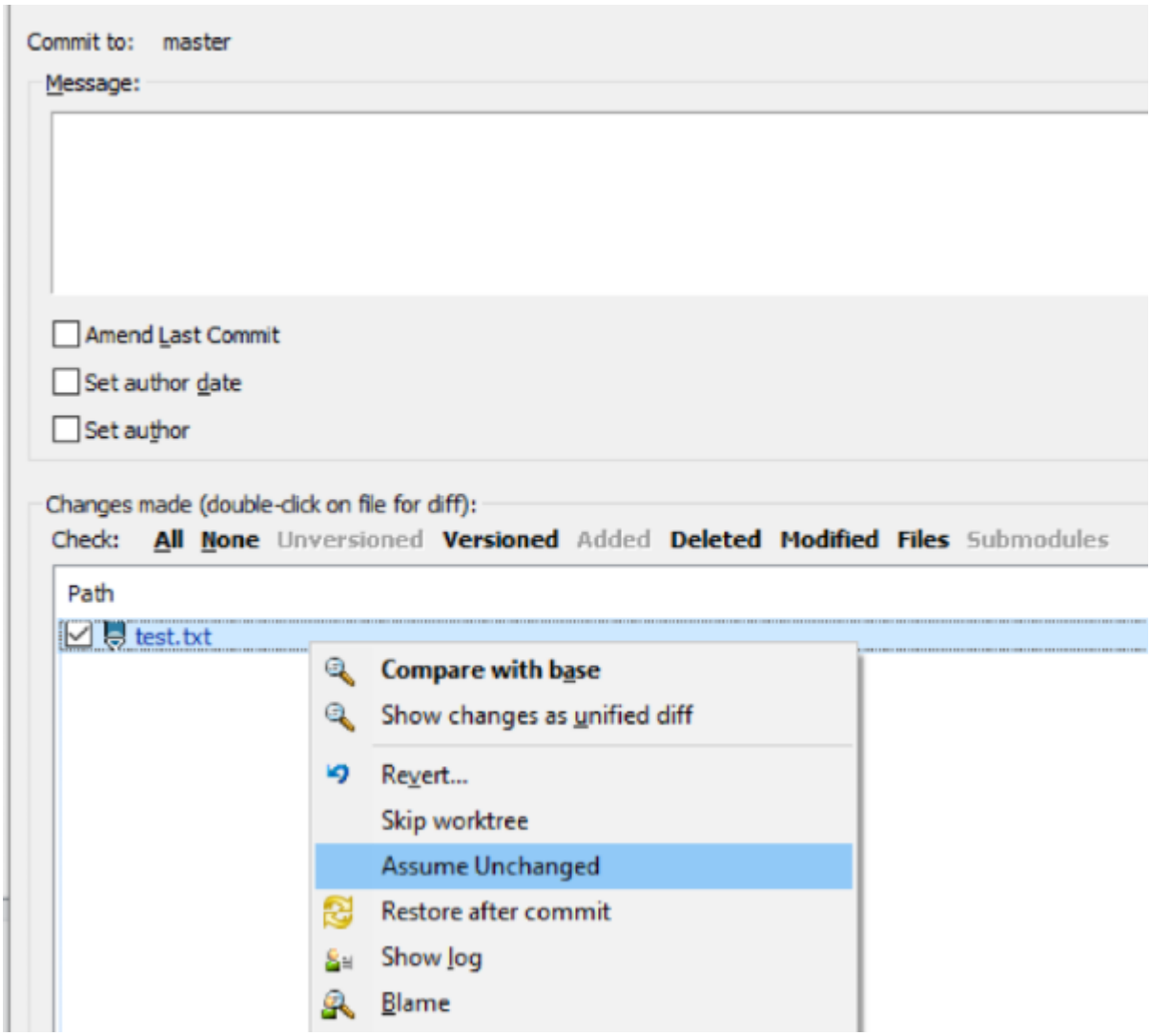


しいウィンドウがきます - > `Give branch a name` - >ボックスを`Switch to new branch Switch to new branch`にをしたいでしょう。 - > [`OK`をクリックするとです。



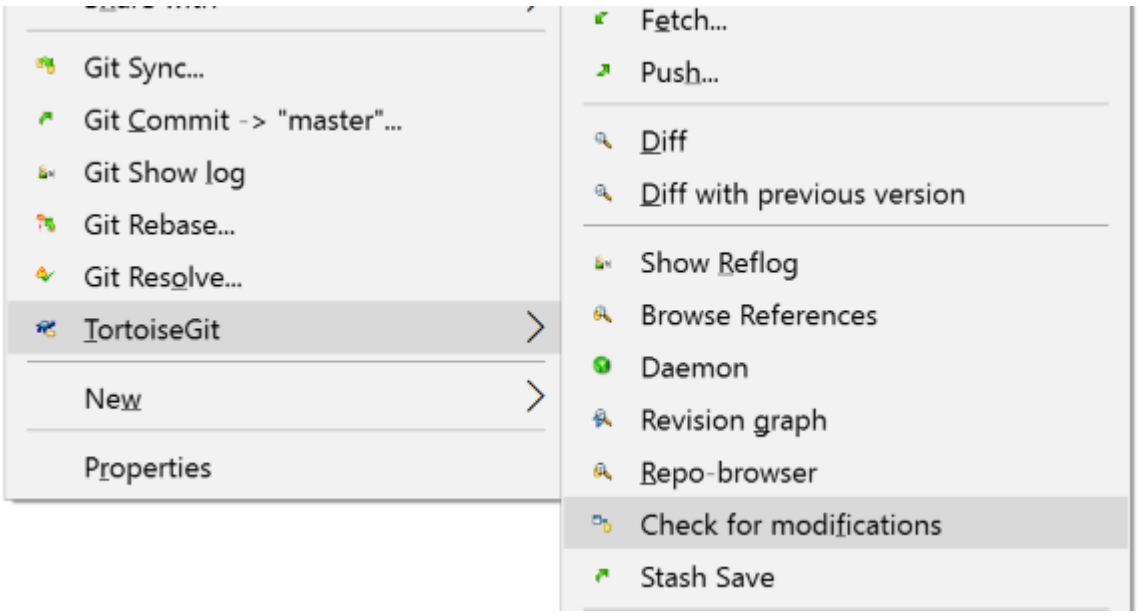
しないとする

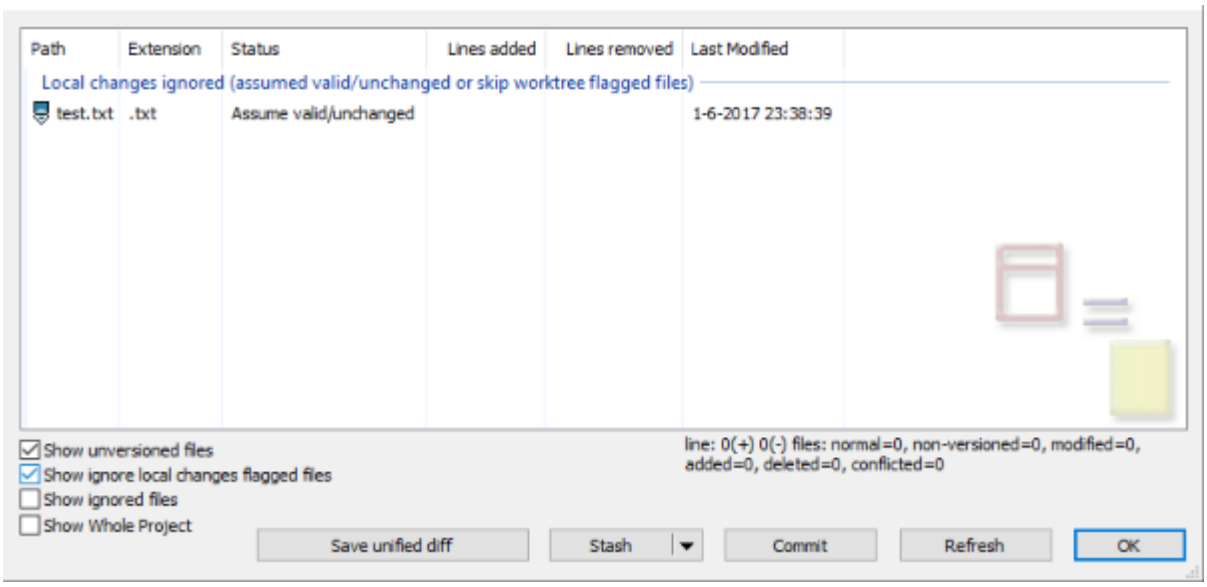
ファイルがされたがコミットしたくないは、ファイルを "しないでください"とします

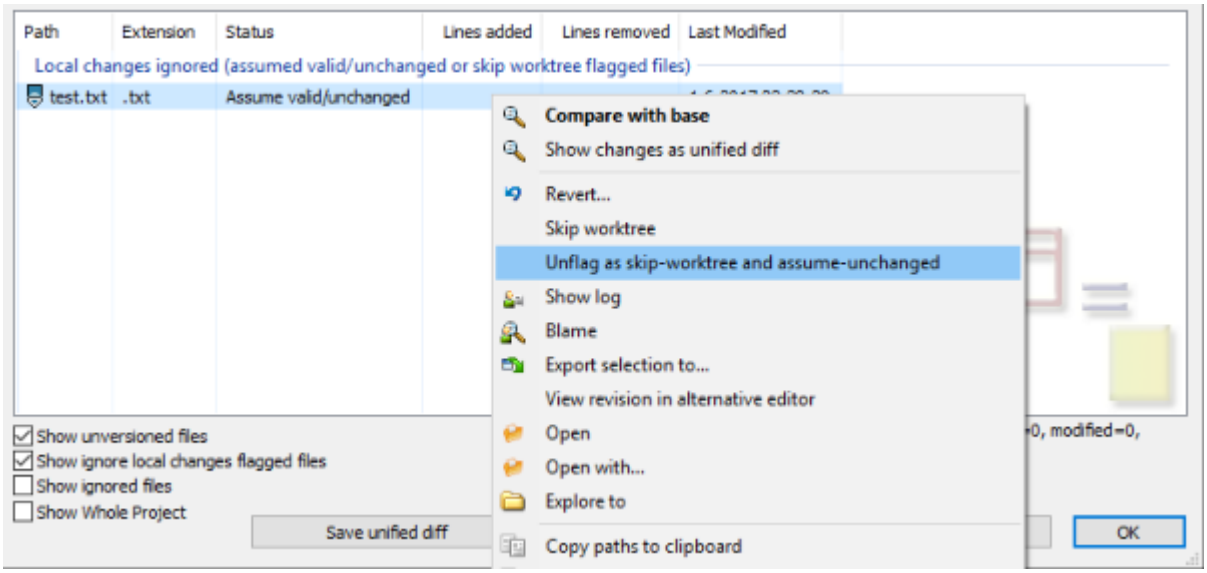


にす

いくつかのステップがです



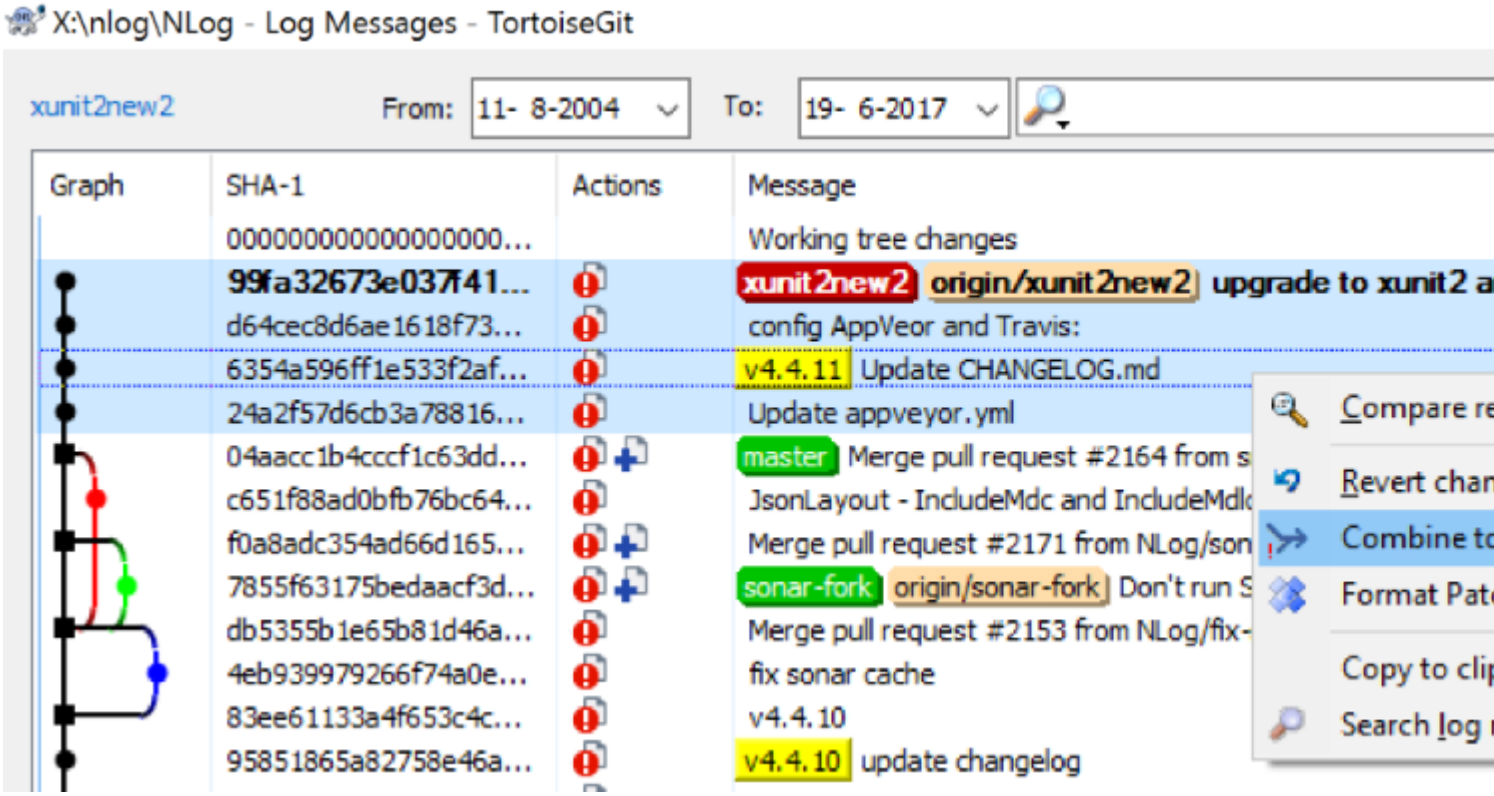




スカッシュがコミットする

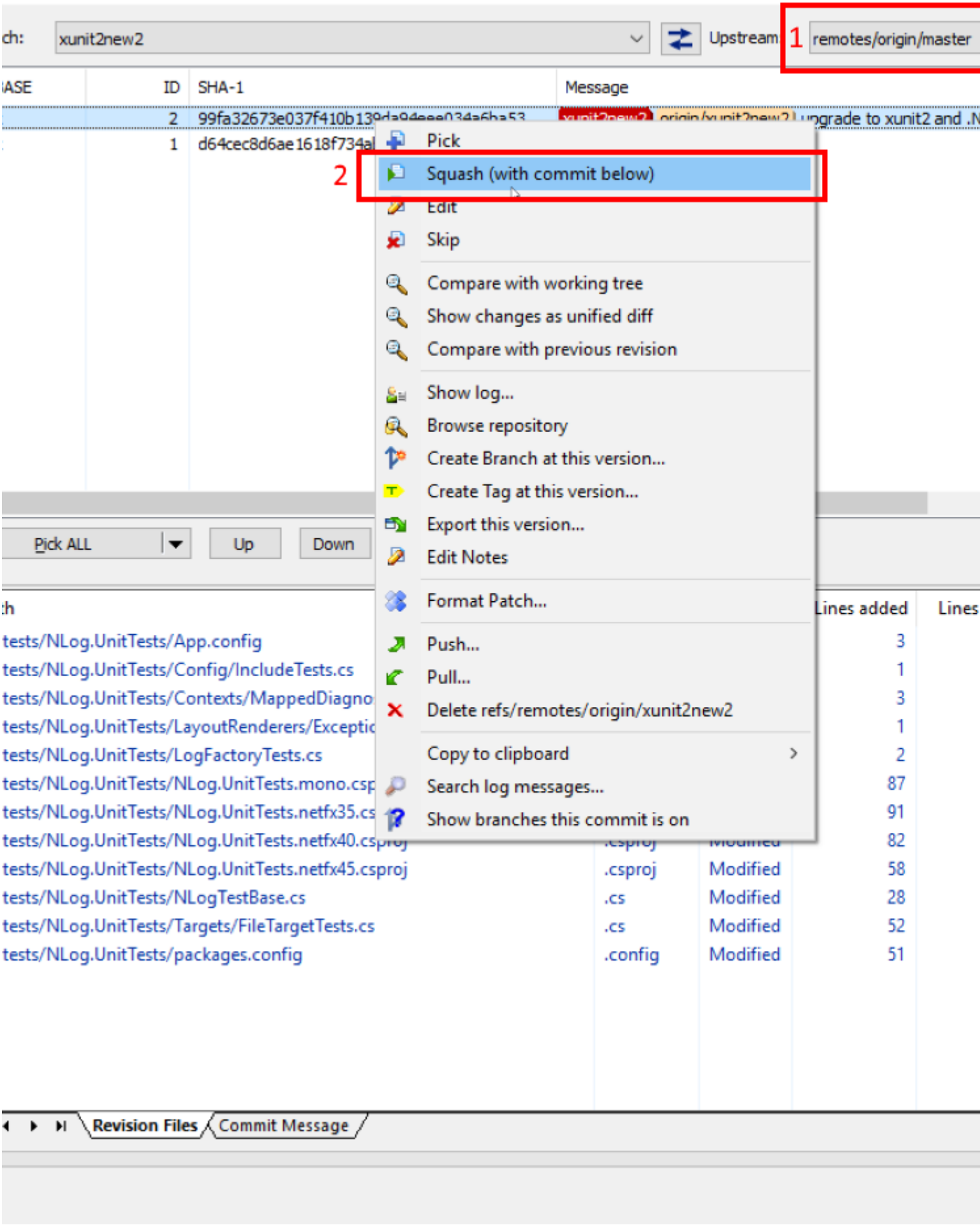
な

あなたのにマージコミットがある、これはしません



な

リベースダイアログをします



オンラインでTortoiseGitをむ https://riptutorial.com/ja/git/topic/5150/tortoisegit

25: アーカイブ

- git archive [--format = <fmt>] [--list] [--prefix = <prefix> /] [<extra>] [-o <file> | --output = <file>] [- worktree-attributes] [--remote = <repo> [--exec = <git-upload-archive>]] <tree-ish> [<path> ...]

パラメーター

パラメータ	
--format = <fmt>	のアーカイブのフォーマット `tar`または`zip`。このオプションがされておらず、ファイルがされている、であればファイルからそのがされます。それの、デフォルトは`tar`ます。
-l、--list	なすべてのをします。
-v、--verbose	をstderrにします。
--prefix = <プレフィックス> /	アーカイブのファイルのに<prefix>をけます。
-o <file>、--output = <file>	アーカイブをstdoutではなく<file>にきします。
--worktree-attributes	ツリーの`.gitattributes`ファイルでを`.gitattributes`ます。
<extra>	これは、アーカイバのバックエンドができるのオプションです。`zip`バックエンドの、`-0`をするとファイルをせずにしますが、`-1 -9`をするととをできます。
--remote = <repo>	ローカルリポジトリではなく、リモートリポジトリ`<repo>`からtarアーカイブをします。
--exec = <git-upload-archive>	リモートの`<git-upload-archive`へのパスをするために`--remote`とともにされます。
<tree-ish>	のアーカイブをするためのツリーまたはコミット。
<path>	オプションのパラメータがなければ、のディレクトリのすべてのファイルとディレクトリがアーカイブにまれます。1つのパスがされているは、これらのみがまれます。

Examples

ディレクトリきのgitリポジトリのア—カイブをする

gitア—カイブをするときにをすると、ディレクトリのすべてのファイルがされます。ディレクトリプレフィックスをつ`HEAD`ア—カイブをするには

```
git archive --output=archive-HEAD.zip --prefix=src-directory-name HEAD
```

すると、のディレクトリの`src-directory-name`というディレクトリのすべてのファイルがされます。

のブランチ、リビジョン、タグ、またはディレクトリについてgitリポジトリのア—カイブをする

ブランチ、コミット、タグ、ディレクトリなど、 `HEAD`のアイテムのア—カイブをすることもできます。

ロ—カルブランチ`dev`ア—カイブをするには

```
git archive --output=archive-dev.zip --prefix=src-directory-name dev
```

`origin/dev`リモ—トブランチのア—カイブをするには

```
git archive --output=archive-dev.zip --prefix=src-directory-name origin/dev
```

タグ`v.01`ア—カイブをするには

```
git archive --output=archive-v.01.zip --prefix=src-directory-name v.01
```

リビジョン`HEAD`のサブディレクトリ `sub-dir` にファイルのア—カイブをする

```
git archive zip --output=archive-sub-dir.zip --prefix=src-directory-name HEAD:sub-dir/
```

gitリポジトリのア—カイブをする

`git archive`をすると、リポジトリのア—カイブをすることができますえば、リリ—スをする。

の`HEAD`リビジョンのtarア—カイブをします。

```
git archive --format tar HEAD | cat > archive-HEAD.tar
```

の`HEAD`リビジョンのtarア—カイブをgzipでします

```
git archive --format tar HEAD | gzip > archive-HEAD.tar.gz
```

これは、みみのtar.gzのをするのでもうことができます。

```
git archive --format tar.gz HEAD > archive-HEAD.tar.gz
```

の`HEAD`リビジョンのzipアーカイブをする

```
git archive --format zip HEAD > archive-HEAD.zip
```

わりに、なをつファイルをするだけで、フォーマットとタイプがされます。

```
git archive --output=archive-HEAD.tar.gz HEAD
```

オンラインでアーカイブをむ https://riptutorial.com/ja/git/topic/2815/アーカイブ

26: あなたのローカルとリモートのリポジトリをする

Examples

リモートでされたローカルブランチをする

ローカルとされたリモートブランチのリモートトラッキングには、

```
git fetch -p
```

あなたはそれをすることができます

```
git branch -vv
```

どのブランチがもはやトラッキングされていないことをします。

もはやされていないブランチは、のフォームになります。

```
 branch              12345e6 [origin/branch: gone] Fixed bug
```

のコマンドのみわせをして、 'git branch -vv'が 'gone'をし、に '-d'をってブランチをするをします

```
git fetch -p && git branch -vv | awk '/: gone]/{print $1}' | xargs git branch -d
```

オンラインであなたのローカルとリモートのリポジトリをするをむ
https://riptutorial.com/ja/git/topic/10934/あなたのローカルとリモートのリポジトリをする

27: エイリアス

Examples

シンプルエイリアス

Gitにエイリアスをするは2つあります。

- `~/.gitconfig`ファイルを`~/.gitconfig`

```
[alias]
    ci = commit
    st = status
    co = checkout
```

- コマンドラインで

```
git config --global alias.ci "commit"
git config --global alias.st "status"
git config --global alias.co "checkout"
```

エイリアスがされた、のようにします。

- `git ci` `git commit`わりに`git ci`、
- `git st`わりに`git st`を`git status`。
- `git co` `git checkout`わりに`git co`を`git checkout`ます。

のgitコマンドとに、エイリアスはのにできます。えば

```
git ci -m "Commit message..."
git co -b feature-42
```

のエイリアスの/

のgitエイリアスを`--get-regexp`をってリストすることができます

```
$ git config --get-regexp '^alias\.'
```

エイリアスの

エイリアスをするには、`[alias] .gitconfig`のをします。

```
aliases = !git config --list | grep ^alias\\. | cut -c 7- | grep -Ei --color \"$1\" "#"
```

に、

- `git aliases` - すべてのエイリアスをする
- `git aliases commit` - "commit"をむエイリアスのみ

なエイリアス

Gitをうと、にGitのコマンドと`sh`をすることができます`!`。

あなたの`~/.gitconfig`ファイルで

```
[alias]
    temp = !git add -A && git commit -m "Temp"
```

これらのプレフィックスきエイリアスでフルシェルができるということは、シェルをして、コマンドラインをするようなよりなエイリアスをできることをします。

```
[alias]
    ignore = "!f() { echo $1 >> .gitignore; }; f"
```

のエイリアスは`f`をし、エイリアスにすでそれをします。だから、している`git ignore .tmp/`します`.tmp/`あなたに`.gitignore`ファイル。

、このパターンはにで、Gitは`$1` 、 `$2`などのをしているので、なをするはありません。 ただし、これらのをってアクセスしても、Gitはをするので、にダミーコマンドをすることをおめします。

`!`にエイリアスがあることにしてください`!`のディレクトリがツリーでいでも、git checkoutのルートディレクトリからされます。これはにそこに`cd`するなしにルートからコマンドをするのになです。

```
[alias]
    ignore = "! echo $1 >> .gitignore"
```

トラッキングされたファイルをにする

にファイルをしてマークするエイリアスへのパラメータとしてファイルをす - タイプ

```
unwatch = update-index --assume-unchanged
```

ファイルのをするには -

```
watch = update-index --no-assume-unchanged
```

にされたすべてのファイルをするには、のようにします。

```
unwatched = "!git ls-files -v | grep '^[[:lower:]]'"
```

のリストをするには、のようにします。

```
watchall = "!git unwatched | xargs -L 1 -I % sh -c 'git watch `echo % | cut -c 2-`'"
```

エィリアスの

```
git unwatch my_file.txt
git watch my_file.txt
git unwatched
git watchall
```

ブランチグラフをったきれいなログをする

```
[alias]
  logp=log --pretty=format:'%h %ad | %s%d [%an]' --graph --date=short

  lg =  log --graph --date-order --first-parent \
     --pretty=format:'%C(auto)%h%Creset %C(auto)%d%Creset %s %C(green)(%ad) %C(bold
cyan)<%an>%Creset'
  lgb = log --graph --date-order --branches --first-parent \
     --pretty=format:'%C(auto)%h%Creset %C(auto)%d%Creset %s %C(green)(%ad) %C(bold
cyan)<%an>%Creset'
 lga = log --graph --date-order --all \
   --pretty=format:'%C(auto)%h%Creset %C(auto)%d%Creset %s %C(green)(%ad) %C(bold
cyan)<%an>%Creset'
```

ここでは、-- `--pretty`でされるオプションとプレ―スホルダのなリストは`git help log`です

--graph - コミットツリ―をする

--date-order - できるだけコミットタイムスタンプをする

--first-parent - マ―ジノ―ドののにいます。

--branches - すべてのロ―カルブランチをしますデフォルトでは、のブランチのみがされます

--all - すべてのロ―カルとリモ―トのブランチをする

h - コミットのハッシュ

ad - スタンプ

an - のユ―ザ―

an - ユ―ザ―をコミットする

Cauto - [color]セクションでされたをする

Creset - をリセットする

d - - デコレ―トブランチとタグ

s - メッセージをコミットする

ad - の--dateのにいますコミッタのではありません

an - コミッターのcn

をしながらコードをする

によっては、コードのしないをコミットするがあります。ブランチでしばらくしているは、の`git pull`をうがある、とのマージをするので、これはしいかもしれません。

```
[alias]
  up = pull --rebase
```

これはあなたのアップストリームソースでされ、あなたがプルダウンしたもののにプッシュしていないをします。

するには

```
git up
```

あなたの.gitignoreでされるファイルをする

```
[ alias ]

    ignored = ! git ls-files --others --ignored --exclude-standard --directory \
            && git ls-files --others -i --exclude-standard
```

ファイルごとに1をするのでgrepディレクトリのみ

```
$ git ignored | grep '/$'
.yardoc/
doc/
```

またはカウント

```
~$ git ignored | wc -l
199811                # oops, my home directory is getting crowded
```

ステージングされたファイルをステージングする

、 `git reset` commitをしてコミットされるようにステージングされたファイルをするには、 `reset` えられたにじてくのがあります。にunstageすべてのファイルは、たちがしてしいエイリアスをするためのgitのをすることができ、な`reset`が、々がするためにしいをするために、えておくはありません`reset`。

```
git config --global alias.unstage "reset --"
```

、あなたがステージファイルをステージングしたくないときはいつでも、 `git unstage`とタイプすればいいです。

オンラインでエイリアスをむ https://riptutorial.com/ja/git/topic/337/エイリアス

28: コミット

き

Gitとのコミットは、コーードのをにさせることによってをします。 Gitは、コミットのとセキュリティのためののをします。このトピックでは、Gitをしてコミットするのなプラクティスとについてします。

- git commit [flags]

パラ メーター

パラ メータ	
- メッセージ、-m	コミットにめるメッセージ。このパラメータをすると、エディタをくというGitののがバイパスされます。
--amend	ステージングされているをのコミットにするようにします。して、これはをきえることができます
--no-edit	エディタをせずに、したコミットメッセージをします。たとえば、 `git commit --amend --no-edit`はコミットメッセージをせずにコミットをします。
--all、 - a	まだステージングされていないをむ、すべてのをします。
-	コミットにするをでします。
- のみ	されたパスのみをコミットします。これは、そうしないり、あなたがっているものをコミットしません。
--patch、-p	パッチインタフェースをして、どのをコミットするかをします。
- けて	`git commit` manページをします。
-S [keyid]、-S --gpg-sign [= keyid]、-S --no-gpg-sign	コミットのサイン、GPGサインのコミット、countermand `commit.gpgSign`
-n、--no-verify	このオプションは、コミットコミットとフックをバイパスします。 フックもしてください

Examples

エディタをかずにコミットする

Gitは、 `git commit`をするときに `vim`や`emacs`ようなエディタをきます。コマンドラインからメッセージをするには、 `-m`オプションをします。

```
git commit -m "Commit message here"
```

コミットメッセージはのにまたがることができます

```
git commit -m "Commit 'subject line' message here

More detailed description follows here (after a blank line)."
```

あるいは、の`-m`をすこともできます。

```
git commit -m "Commit summary" -m "More detailed description follows here"
```

「*Git Commit Messageをく」*をしてください。

*Udacity Git Commit*メッセージスタイルガイド

コミットの

のコミットがまだされていないのリポジトリにプッシュされていない、コミットをすることができます。

```
git commit --amend
```

これは、のなをのコミットにきます。

これは、ったコミットメッセージをするためにもできます。これは、デフォルトのエディタは`vi` / `vim` / `emacs` をし、のメッセージをできるようにします。

コミットメッセージをインラインでするには

```
git commit --amend -m "New commit message"
```

のコミット･メッセージをせずにするには、のようにします。

```
git commit --amend --no-edit
```

するとコミットのはされますが、のはされません。をするようgitにできます。

```
git commit --amend --reset-author
```

コミットのをすることもできます

```
git commit --amend --author "New Author <email@address.com>"
```

のコミットをするとにきえられ、のコミットはブランチのからされることにしてください。これは、リポジトリやのとのブランチでするにしてください。

つまり、のコミットがすでにプッシュされていたは、それをしてからに`push --force`するがあります。

をコミットする

、 `git add`または`git rm`をして、 `git commit`するにをインデックスにするがあります。 `-a`または`--all`オプションをすと、されたファイルへのすべてのを、をむインデックスににできます。

```
git commit -a
```

コミットメッセ―ジをするは、のようにします。

```
git commit -a -m "your commit message goes here"
```

また、2つのフラグをすることもできます。

```
git commit -am "your commit message goes here"
```

すべてのファイルをにコミットするはありません。 `-a`または`--all`フラグをし、コミットするファイルをします。

```
git commit path/to/a/file -m "your commit message goes here"
```

ののファイルをコミットするには、1つまたはのファイル、ディレクトリ、パタ―ンもできます。

```
git commit path/to/a/file path/to/a/folder/* path/to/b/file -m "your commit message goes here"
```

のコミットの

に、のコミットまたはとののコミットはエラ―です。

しかし、ビルド･フック、CIシステム、およびコミットをきこすのシステムをテストする、ダミ―･ファイルを/タッチすることなくコミットをにできるとです。

`--allow-empty`コミットはチェックをバイパスします。

```
git commit -m "This is a blank commit" --allow-empty
```

ステ―ジングとのコミット

あなたがそれらをコミットするに、あなたのソースコードにをえた、あなたはGitリポジトリをつそれらのをステージングするがあります。

たとえば、 `README.md`と`program.py`をしたとします`program.py`

```
git add README.md program.py
```

これはgitにのコミットにファイルをしたいことをえます。

に、

```
git commit
```

これはテキストエディタをきますが、それはしばしば vimです。 vimにしていないは、 `i`をしてモードにり、コミットメッセージをき、 `Esc`と`:wq`を`Esc`てしてすることができます。テキストエディタをくのをけるには、メッセージに`-m`フラグをけるだけです

```
git commit -m "Commit message here"
```

コミットメッセージは、しばしばのフォーマットルールにいます。については、「なコミットメッセージ」をしてください。

ショートカット

ディレクトリのくのファイルをしたは、それぞれのファイルをするのではなく、のコマンドをできます。

```
git add --all        # equivalent to "git add -a"
```

または、 されたファイルをく、すべてのをトップレベルのディレクトリとサブディレクトリからするには、のようにします 。

```
git add .
```

または、されているファイルのみをする「 」

```
git add -u
```

にじて、なをします。

```
git status           # display a list of changed files
git diff --cached    # shows staged changes inside staged files
```

に、をコミットします。

```
git commit -m "Commit message here"
```

わりに、のファイルやされたファイルのみをし、しいファイルをしていないは、`git add`と`git commit`アクションを1つのコマンドでみわせることができます

```
git commit -am "Commit message here"
```

これは`git add --all`とじですべてのされたファイルをステージングすることにしてください。

データ

パスワードやなどのデータをしてしててるべきではありません。このケースがし、がすでにサーバーにプッシュされているは、データがされたものとなします。それのは、でそのデータをすることができます。かつなは、「BFG Repo-Cleaner」https://rtyley.github.io/bfg-repo-cleaner/ のです。

コマンド`bfg --replace-text passwords.txt my-repo.git`は`passwords.txt`ファイルから`passwords.txt`みみ、`***REMOVED***`きえます。このでは、リポジトリののコミットをすべてします。

かのためにコミットする

のかがあなたがコミットしているコードをいたなら、あなたは`--author`オプションをけてクレジットをえることができます

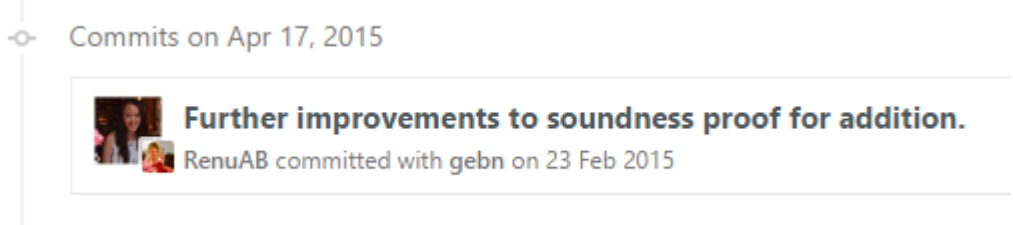
```
git commit -m "msg" --author "John Smith <johnsmith@example.com>"
```

Gitがのをするためにするパターンをすることもできます

```
git commit -m "msg" --author "John"
```

この、"John"をむとのものコミットのがされます。

GitHubでは、のいずれかのでされたコミットは、きなのサムネイルをします。



のファイルのをコミットする

のファイルにえられたをコミットし、 `git add`をってステーージングをスキップすることができます

```
git commit file1.c file2.h
```

または、ファイルをにステーージングすることもできます。

```
git add file1.c file2.h
```

でコミットしてください

```
git commit
```

いコミットメッセーージ

`git log`をたどるが、それぞれのコミットがであるかをにすることがです。なコミットメッセーージには、、トラッカーのタスクまたはのと、されたと、およびによってはそれがどのようにわれたかにするながまれています。

よりいメッセーージはのようになります。

```
TASK-123: Implement login through OAuth
TASK-124: Add auto minification of JS/CSS files
TASK-125: Fix minifier error when name > 200 chars
```

のメッセーージはあまりにちませんが、

```
fix                          // What has been fixed?
just a bit of a change       // What has changed?
TASK-371                     // No description at all, reader will need to look at the tracker
themselves for an explanation
Implemented IFoo in IBar     // Why it was needed?
```

コミット･メッセーージがしいでかれているかどうかをテストするは、ブランクをメッセーージにきえて、があるかどうかをすることです。

このコミットをすると、はのリポジトリに___します。

らしいgit commitメッセーージの7つのルーール

1. とをのでります
2. を50にする
3. をにする
4. にピリオドをつけないでください
5. にのをする
6. のを72でりします。
7.

をっての でするのか

Chris Beamのブログの7つのルール 。

のにコミットする

```
git commit -m 'Fix UI bug' --date 2016-07-01
```

`--date`パラメータは、 のをします 。 このは、たとえば`git log`にされ`git log`。

コミットもするには

```
GIT_COMMITTER_DATE=2016-07-01 git commit -m 'Fix UI bug' --date 2016-07-01
```

dateパラメータは、GNUのでサポートされているようななをけれます。

```
git commit -m 'Fix UI bug' --date yesterday
git commit -m 'Fix UI bug' --date '3 days ago'
git commit -m 'Fix UI bug' --date '3 hours ago'
```

がをしていない、のがされ、のみがオーバーライドされます。

コミットのためにステージングするをする

1つまたはのファイルにくのがえられているとしますが、をコミットするファイルごとに、のようにすることができます。

```
git add -p
```

または

```
git add -p [file]
```

それぞれのがにされ、ごとにのオプションのいずれかをするようにめられます。

```
y - Yes, add this hunk

n - No, don't add this hunk

d - No, don't add this hunk, or any other remaining hunks for this file.
    Useful if you've already added what you want to, and want to skip over the rest.

s - Split the hunk into smaller hunks, if possible

e - Manually edit the hunk.  This is probably the most powerful option.
    It will open the hunk in a text editor and you can edit it as needed.
```

これにより、したファイルのがステージングされます。に、このようななをすべてコミットでき

ます。

```
git commit -m 'Commit Message'
```

ステ―ジングまたはコミットされなかったは、ききファイルにされ、にじてでコミットできます。または、りのがましくないは、のでできます。

```
git reset --hard
```

よりさなコミットにきくすることをにすれば、このアプロ―チはあなたがコミットしようとしているものをすのにもちます。それぞれのをにすることで、あなたがいたものをチェックするがあり、println / loggingのようなましくないコ―ドをってステ―ジングすることをけることができます。

コミットのの

コミットのをするには

```
git commit --amend --date="Thu Jul 28 11:30 2016 -0400"
```

あるいは

```
git commit --amend --date="now"
```

コミットのをする

ったとしてコミットした、それをしてからすることができます

```
git config user.name "Full Name"
git config user.email "email@example.com"

git commit --amend --reset-author
```

GPGがコミットする

1. あなたののIDをする

   ```
   gpg --list-secret-keys --keyid-format LONG

   /Users/davidcondrey/.gnupg/secring.gpg
   ------------------------------------
   sec   2048R/YOUR-16-DIGIT-KEY-ID YYYY-MM-DD [expires: YYYY-MM-DD]
   ```

   あなたのIDは、のスラッシュにくの16のコ―ドです。

2. あなたのgitのであなたののIDをする

```
git config --global user.signingkey YOUR-16-DIGIT-KEY-ID
```

3. バージョン1.7.9、git commitはコミットにをける-Sオプションをけれます。このオプションをすると、GPGパスフレーズがされ、コミットログにがされます。

```
git commit -S -m "Your commit message"
```

オンラインでコミットをむ https://riptutorial.com/ja/git/topic/323/コミット

29: さくらんぼり

き

チェリ—ピックは、コミットでされたパッチをりし、あなたがいるブランチにしようとします。

Git SCM Book

- git cherry-pick [-edit] [-n] [-m] [-s] [-x] [--ff] [-S [key-id]]コミット...
- git cherry-pick -
- git cherry-pick --quit
- git cherry-pick --abort

パラ メ—タ—

パラ メ—タ—	
-e、--edit	このオプションをすると、 `git cherry-pick`はコミットするにコミットメッセ—ジをできるようになります。
-バツ	コミットをするとき、のコミットメッセ—ジに "チェリ—はコミットからんだ..."というをして、このがチェリ—ピックされたことをします。これはのないチェリ—ピックのみにわれます。
--ff	のHEADがcherry-pick'edコミットのとじ、このコミットへのりがされます。
- する	.git / sequencerのをしてのをします。したチェリ—ピックまたはのをしたもすることができます。
- する	ののについてはれてください。チェリ—ピックまたはにしたにシ—ケンサ—のをクリアするためにできます。
- アボ—ト	をキャンセルし、シ—ケンスのにります。

Examples

あるブランチからのブランチへのコミットのコピ—

`git cherry-pick <commit-hash>`は、のコミットのをのブランチにし、しいコミットをします。に、ブランチからブランチにコミットをコピ—できます。

えられた

```
dd2e86 - 946992 - 9143a9 - a6fd86 - 5a6057 [master]
           \
            76cada - 62ecb3 - b886a0 [feature]
```

`b886a0`をマスターにコピーしたいとしましょう `5a6057`に。

たちはれる

```
git checkout master
git cherry-pick b886a0
```

、たちのはのようになります

```
dd2e86 - 946992 - 9143a9 - a6fd86 - 5a6057 - a66b23 [master]
           \
            76cada - 62ecb3 - b886a0 [feature]
```

しいコミット`a66b23`が`a66b23`とじソース、コミットメッセージを`b886a0` ただし、の。チェリーピッキングは、そのブランチのこのは`b886a0` のみをピックアップすることにしてくださいこのためには、 `b886a0`またはマージをするがあります。

1つのブランチからのブランチにコミットのをコピーする

`git cherry-pick <commit-A>..<commit-B>`は、 Aののすべてのコミットを、チェックアウトされているブランチのにBをめてします。

`git cherry-pick <commit-A>^..<commit-B>`はコミットAと、チェックアウトされているブランチのにBをむすべてのコミットをします。

チェリーピックがかどうかの

チェリー・ピック・プロセスをするに、チェリー・ピックするコミットがターゲット・ブランチにすでにするかどうかをできます。その、もするはありません。

`git branch --contains <commit>`は、されたコミットをむローカルブランチをリストします。

`git branch -r --contains <commit>`は、リストのリモートブランチもまれます。

にまだされていないコミットをつける

コマンド`git cherry`はまだチェリーピックされていないをします。

```
git checkout master
git cherry development
```

...とがしこのようにされます

```
+ 492508acab7b454eee8b805f8ba906056eede0ff
- 5ceb5a9077ddb9e78b1e8f24bfc70e674c627949
+ b4459544c000f4d51d1ec23f279d9cdb19c1d32b
+ b6ce3b78e938644a293b2dd2a15b2fecb1b54cd9
```

にいることをコミット`+`まだにみしていないものになります`development`。

```
git cherry [-v] [<upstream> [<head> [<limit>]]]
```

オプション

-v SHA1のにコミットをします。

<upstream>のコミットをするブランチ。デフォルトはHEADのブランチです。

<head>ブランチ。デフォルトはHEADです。

<limit>コミットをおよびそれをむとしてしないでください。

については、 git-cherryのドキュメントをしてください。

オンラインでさくらんぼりをむ https://riptutorial.com/ja/git/topic/672/さくらんぼり

30: サブツリー

- `git subtree add -P <prefix> <commit>`
- `git subtree add -P <prefix> <repository> <ref>`
- `git subtree pull -P <prefix> <repository> <ref>`
- `git subtree push -P <prefix> <repository> <ref>`
- `git subtree merge -P <prefix> <commit>`
- `git subtree split -P <prefix> [OPTIONS] [<commit>]`

これは、 `submodule`をするわりのです

Examples

サブツリーの、プル、バックポート

サブツリーの

プラグインのリポジトリをす`plugin`とばれるしいリモートをします

```
git remote add plugin https://path.to/remotes/plugin.git
```

に、しいフォルダプレフィックス`plugins/demo`するサブツリーをします。 `plugin`はリモートで、`master`はサブツリーのリポジトリのmasterブランチをします。

```
git subtree add --prefix=plugins/demo plugin master
```

サブツリーのをきす

プラグインでわれたのコミットをきす

```
git subtree pull --prefix=plugins/demo plugin master
```

バックポートサブツリー

1. スーパープロジェクトでバックポルトするコミットをする

   ```
   git commit -am "new changes to be backported"
   ```

2. マージのためにしいブランチをチェックアウトし、サブツリーリポジトリをするようにする

```
git checkout -b backport plugin/master
```

3. チェリーピックバックポート

```
git cherry-pick -x --strategy=subtree master
```

4. をプラグインソースにす

```
git push plugin backport:master
```

オンラインでサブツリーをむ https://riptutorial.com/ja/git/topic/1634/サブツリー

31: サブモジュール

Examples

サブモジュールの

のGitリポジトリを、Gitによってされるプロジェクトのフォルダとしてめることができます

```
$ git submodule add https://github.com/jquery/jquery.git
```

しい`.gitmodules`ファイルをしてコミットするがあります。 `git submodule update`がされたときに、どのサブモジュールをクローンするべきかをGitにえます。

サブモジュールをつGitリポジトリのクローン

サブモジュールをするリポジトリをクローンするときは、それらをしてするがあります。

```
$ git clone --recursive https://github.com/username/repo.git
```

これにより、されたサブモジュールがクローンされ、なフォルダサブモジュールのサブモジュールをむにされます。これは、クローンがしたに`git submodule update --init --recursive`をするのとじです。

サブモジュールの

サブモジュールはのリポジトリののコミットをします。すべてのサブモジュールでされているなをチェックするには、

```
git submodule update --recursive
```

によっては、されているをするわりに、ローカルのチェックアウトをリモートのサブモジュールののにするがあります。 1つのコマンドですべてのサブモジュールをリモートののにチェックアウトするには、

```
git submodule foreach git pull <remote> <branch>
```

デフォルトの`git pull`をする

```
git submodule foreach git pull
```

これはローカルコピーをするだけであることにしてください。 `git status`をすると、このコマンドのためにサブモジュールディレクトリがされた、そのサブモジュールディレクトリがdirtyとされます。わりにしいをするようにリポジトリをするには、をコミットするがあります。

```
git add <submodule_directory>
git commit
```

`git pull`をしている、 `git pull --rebase`をしてをにすことができます。ほとんどの、のがくなります。また、すべてのブランチをロ―カルにきします。

```
git submodule foreach git pull --rebase
```

のサブモジュ―ルののをチェックアウトするには、をできます。

```
git submodule update --remote <submodule_directory>
```

サブモジュ―ルをブランチにうようにする

サブモジュ―ルはに、のコミットSHA1 "gitlink"、リポジトリのインデックスのなエントリでチェックアウトされます。

しかし、サブモジュ―ルリモ―トリポジトリのブランチののコミットにそのサブモジュ―ルをするようすることができます。

サブモジュ―ルにくのではなく、 `git checkout abranch --track origin/abranch`, `git pull`をすると、リポジトリからにうことができます。

```
git submodule update --remote --recursive
```

サブモジュ―ルのSHA1がされるので、あなたはそれにくがあります

```
git add .
git commit -m "update submodules"
```

それはサブモジュ―ルがのものだったとします

- フォロ―するブランチをするか、

  ```
  git submodule -b abranch -- /url/of/submodule/repo
  ```

- のサブモジュ―ルのためにブランチにうようにすることができます。

  ```
  cd /path/to/parent/repo
  git config -f .gitmodules submodule.asubmodule.branch abranch
  ```

サブモジュ―ルの

1.8

サブモジュ―ル `the_submodule` をびすには

```
$ git submodule deinit the_submodule
$ git rm the_submodule
```

- `git submodule deinit the_submodule`は.git / configから`the_submodule`のエントリをします。これはthe_submoduleを`git submodule update` 、 `git submodule sync` 、 `git submodule foreach`びしからし、ローカルコンテンツソースをします。また、これはリポジトリにとしてされません。 `git submodule init`と`git submodule update`は、リポジトリにコミットなをえずに、サブモジュールをします。

- `git rm the_submodule`は、サブモジュールをツリーからします。ファイルは`.gitmodules`ファイルソースのサブモジュールのエントリとになくなります。ただし、 `git rm the_submodule` の`git submodule deinit the_submodule`がされていないは、.git / configファイルのサブモジュールのエントリはそのまります。

1.8

ここから

1. `.gitmodules`ファイルからするセクションをします。
2. ステージを`.gitmodulesgit add .gitmodules`
3. `.git/config`からするセクションをします。
4. `git rm --cached path_to_submodule`しますにスラッシュはありません。
5. `rm -rf .git/modules/path_to_submodule`します。
6. Commit `git commit -m "Removed submodule <name>"`
7. のサブモジュールファイルをする
8. `rm -rf path_to_submodule`

サブモジュールの

1.8

```
$ git mv old/path/to/module new/path/to/module
```

`1.8`

1. `.gitmodules`を□□し、サブモジュールのパスを□□に□□し、 `git add .gitmodules`□ってインデックスに□れて`git add .gitmodules`。
2. □□に□じて、サブモジュールの□しい□□の□ディレクトリを□□します□ `mkdir -p new/path/to` □。
3. すべてのコンテンツを□いディレクトリから□しいディレクトリに□□します□ `mv -vi old/path/to/module new/path/to/submodule` □。
4. Gitがこのディレクトリを□□していることを□□してください□ `git add new/path /to` □。
5. `git rm --cached old/path/to/module`□いディレクトリを□□し`git rm --cached old/path/to/module`。
6. すべての□□を`.git/modules/ new/path/to/module .git/modules/ old/path/to/module`、ディレクトリ`.git/modules/ old/path/to/module`□□し`.git/modules/ new/path/to/module`。
7. `.git/modules/ new/path/to /config`ファイルを□□し、`worktree`□□が□しい□□を□していることを□□してください。この□では`worktree = ../../../../../ old/path/to/module`。□□□には、その□□の□□□□に2つ□□のディレクトリが□□するはず`..`。。 `new/path/to/module /.git`ファイルを□□し、そのパスがメインプロジェクトの`.git`フォ

ルダ□の□しい□□を□していることを□□します。この□では`gitdir: ../../../.git/modules/new/path/to/module`。

`git status`□□は□で□のようになります□

```
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   .gitmodules
#       renamed:    old/path/to/submodule -> new/path/to/submodule
#
```

8. □□に、□□をコミットします。

Axel BeckertによるStack Overflowのこの□

オンラインでサブモジュールを□む□ https://riptutorial.com/ja/git/topic/306/サブモジュール

32: ジットパッチ

□□

- `git am [--signoff] [--keep] [- [no-] keep-cr] [- [no-] utf8] [- 3way] [--interactive] [--committer-date-is -author-date] [--ignore-date] [--ignore-space-change | --ignore-whitespace] [--whitespace = <option>] [-C <n>] [-p <n>] [--directory = <dir>] [--exclude = <path>] [- include = <path>] [--reject] [-q | [- [no -]はさみ] [-S [<keyid>]] [--patch-format = <format>] [□<mbox> | <Maildir>□...]`
- `git am□--continue | --skip | --abort□`

パラメーター

パラメータ	
<mbox> | <Maildir>...	パッチをみむメールボックスファイルのリスト。このをしないと、コマンドはからみみます。ディレクトリをすると、それらはMaildirとしてわれます。
-s、--signoff	あなたのコミッターIDをして、コミットメッセージにSigned-by-byをします。
-q、--quiet	かにして。エラーメッセージのみをします。
-u、--utf8	`git mailinfo -u`フラグを`git mailinfo`ます。メールからしたコミットログメッセージは、UTF-8エンコーディングにコーディングされています`i18n.commitencoding`をして、UTF-8でないはプロジェクトのエンコーディングをできます。これをにするには`--no-utf8`をできます。
--no-utf8	git mailinfoに-nフラグをします。
-3、 - 3way	パッチがきれいにはされない、パッチがされるとわれるブロブのがされているは3マージにり、ローカルでなブロブをします。
--ignore-date、--ignore-space-change、--ignore-whitespace、--whitespace = <option>、-C <n>、-p <n>、--directory = <dir> exclude = <パス>、--include = <パス>、--reject	これらのフラグはパッチをするgit applyプログラムにされます。
--patch-format	デフォルトでは、このコマンドはパッチフォーマットをにしようとします。このオプションをすると、ユーザーはをバイパスし、パッチをするパッチをできます。なは、 `mbox` 、 `stgit` 、 `stgit-series` 、および`hg`です

パラメータ	
	。
-i、--interactive	にします。
- コミッター - - -	では、コマンドはメールメッセージのをコミットのとしてし、コミットのをコミッターのとしてします。これにより、ユーザーは、とじをしてコミッターについてをつけることができます。
--ignore-date	では、コマンドはメールメッセージのをコミットのとしてし、コミットのをコミッターのとしてします。これにより、ユーザーは、コミッターとじをして、についてをつけることができます。
- スキップ	のパッチをスキップします。これは、されたパッチをするにのみがあります。
-S [<keyid>]、--gpg-sign [=<keyid>]	GPG-signはコミットします。
--continue、-r、--resolved	パッチのえば、するパッチをしようとするの、ユーザはそれをでし、インデックスファイルはアプリケーションのをする。メールメッセージとのインデックスファイルからされたauthorshipとコミットログをしてコミットし、します。
--resolvemsg = <msg>	パッチのがすると、するに`<msg>`がにされます。これにより、のメッセージがきされ、-- `--continue`または`--skip`をしてをするようされます。これは`git rebase`と`git am`でのみにされ`git am`。
- アボート	のブランチをし、パッチをします。

Examples

パッチの□□

パッチを□□するには、2つのステップがあります。

1. □□を□えてコミットします。
2. `git format-patch <commit-reference>`を□□して、`commit <commit-reference>`□それを□まない□からすべてのコミットをパッチファイルに□□します。

たとえば、□□の2つのコミットからパッチを□□する□□は、□のようにします。

```
git format-patch HEAD~~
```

これは、2つのファイルを□□しますHEAD~~□□、コミットごとに1つずつ□□されます。

```
0001-hello_world.patch
0002-beginning.patch
```

パッチの□□

git apply some.patchを□□して、□□の□□ディレクトリに□□された.patchファイルからの□□を□うことができます。□らはステ―ジングされず、コミットする□□があります。

パッチをコミットとして□コミットメッセ―ジとともに□□□するには、

```
git am some.patch
```

すべてのパッチファイルをツリ―に□□するには□

```
git am *.patch
```

オンラインでジットパッチを□む□ https://riptutorial.com/ja/git/topic/4603/ジットパッチ

□ 33: ショー

□□

- `git show [options] <object> ...`

□□

さまざまなGitオブジェクトを□□します。

- コミットについては、コミットメッセージと`diff`を□□します
- タグの□□は、タグメッセージと□□されるオブジェクトを□□します

Examples

□□

`git show`はさまざまなGitオブジェクトを□しています。

コミットの□□□

コミットメッセージと□□された□□の□□を□□します。

コマンド	
`git show`	のコミットをします。
`git show @~3`	から3のコミットをします。

□と□の□□□

ツリーまたはブロブを□□します。

コマンド	
`git show @~3:`	3コミットツリーのプロジェクトルートディレクトリをします。
`git show @~3:src/program.js`	`src/program.js`が3コミットblobであったのをしています
`git show @:a.txt @:b.txt`	のコミットから`a.txt`された`b.txt`します

タグの□□□

タグメッセージと□□されるオブジェクトを□□します。

オンラインでショーを□む□ https://riptutorial.com/ja/git/topic/3030/ショー

34: スクワッシュ

スカッシュって何ですか

Squashは、複数のコミットを取り、複数のコミットからのすべての変更をカプセル化して単一のコミットに圧縮するプロセスです。

スカッシュとリモートブランチ

リモートブランチを追跡しているブランチでコミットを修正するときは、注意してください。既にリモートブランチにプッシュされているコミットをスカッシュすると、2つのブランチが分岐し、`git push -f`を使用してリモートブランチにそれらの変更を強制する必要があります。 これがリモートブランチを操作する人の間に混乱を引き起こす可能性があることに注意してください。そのため、公開されたコミットを共有リポジトリまたは共有リポジトリに既にプッシュするときは注意が必要です。

プロジェクトがGitHub上でホストされている場合は、 `Settings - Branches - Protected Branches`を設定することで、`master`などの特定の分岐で「強制プッシュ」を無効にすることができます。

Examples

リベースなしのスカッシュ最後のコミット

最後のxコミットを単一のものに圧縮する最も簡単な方法は、次のコマンドを使用できます。

```
git reset --soft HEAD~x
git commit
```

xを、squashedコミットに含める最後のコミット数で置き換えます。

これにより、新しいコミットが作成され、作成者、メッセージ、日付を含む最後のxコミットについての情報が編集に入れることに注意してください。おそらく前に最後のコミットメッセージをコピーペーストしたいと思うでしょう。

リベース中に競合することを回避する

コミットは`git rebase`中に押しつぶされる可能性があります。このようにしてコミットを修正しようとする前に、 リベースを実行することをお勧めします。

1. リベースするコミットを特定し、コミットハッシュに注意してください。
2. `git rebase -i [commit hash]`します。

   前の例として、次のようにすることができ`HEAD~4`の代わりに、最後のコミットと4前のコミットを圧縮するために、代わりにコミットハッシュのを。
3. このコマンドを実行するときに開くエディタで、スカッシュするコミットを選択します。置き換えpick次これらの行の先頭に`squash`最後のコミットにそれらをスカッシュします。
4. スカッシュするコミットを選択した後、コミットメッセージを書き込むように求められます。

ロギングは、リベースする前を確認するためにコミットします。

```
> git log --oneline
612f2f7 This commit should not be squashed
d84b05d This commit should be squashed
ac60234 Yet another commit
36d15de Rebase from here
17692d1 Did some more stuff
e647334 Another Commit
```

```
2e30df6 Initial commit

> git rebase -i 36d15de
```

この□□で、□□したエディタがポップアップして、コミットで□をしたいかを□□することができます。 Gitはコメントのヘルプを□□します。あなたがそれをそのままにしておくと、すべてのコミットが□□され、その□□がリベ―スの□と□じになるので、□も□こりません。この□では、□のコマンドを□□します。

```
pick ac60234 Yet another commit
squash d84b05d This commit should be squashed
pick 612f2f7 This commit should not be squashed

# Rebase 36d15de..612f2f7 onto 36d15de (3 command(s))
#
# Commands:
# p, pick = use commit
# r, reword = use commit, but edit the commit message
# e, edit = use commit, but stop for amending
# s, squash = use commit, but meld into previous commit
# f, fixup = like "squash", but discard this commit's log message
# x, exec = run command (the rest of the line) using shell
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
#
# Note that empty commits are commented out
```

コミットメッセ―ジの□き□み□のGitログ

```
> git log --oneline
77393eb This commit should not be squashed
e090a8c Yet another commit
36d15de Rebase from here
17692d1 Did some more stuff
e647334 Another Commit
2e30df6 Initial commit
```

Autosquash□リベ―ス□にスカッシュしたいコ―ドをコミットする

□のような□□を□えると、あなたがコミットしたい□□をしたと□□してくださいbbb2222 A second commit □

```
$ git log --oneline --decorate
ccc3333 (HEAD -> master) A third commit
bbb2222 A second commit
aaa1111 A first commit
9999999 Initial commit
```

□□を□えたら、いつものようにそれらをインデックスに□□して、 --fixup□□を□ってスカッシュするコミットへの□□を□えてコミットすることができます□

```
$ git add .
$ git commit --fixup bbb2222
[my-feature-branch ddd4444] fixup! A second commit
```

これは、Gitがインタラクティブなリベ―ス□に□□できるコミットメッセ―ジで□しいコミットを□□します□

```
$ git log --oneline --decorate
ddd4444 (HEAD -> master) fixup! A second commit
ccc3333 A third commit
bbb2222 A second commit
aaa1111 A first commit
9999999 Initial commit
```

□に、-- --autosquash□□を□□して□□□のリベースを--autosquashます。

```
$ git rebase --autosquash --interactive HEAD~4
```

Gitはあなたが□っコミットスカッシュすることができ□□しますcommit --fixup□しい□□へ□

```
pick aaa1111 A first commit
pick bbb2222 A second commit
fixup ddd4444 fixup! A second commit
pick ccc3333 A third commit
```

すべてのリベースで--autosquashと□□しないようにするには、このオプションをデフォルトで□□にします。

```
$ git config --global rebase.autosquash true
```

マージ□に□□することを□□する

git merge --squashを□□すると、ブランチによって□□された□□を1□のコミットにgit merge --squashせることができます。□□のコミットは□□されません。

```
git merge --squash <branch>
git commit
```

これは、 git resetとほぼ□じですが、□み□まれる□□にシンボリック□が□いている□が□□です。□□□

```
git checkout <branch>
git reset --soft $(git merge-base master <branch>)
git commit
```

□□□□□およびフィックスアップ

□□をコミットするとき、コミットが□□□のコミットに□しつぶされることを□□することが□□であり、そうすることができます。

```
git commit --squash=[commit hash of commit to which this commit will be squashed to]
```

フィックスアップの□わりに、--fixup --fixup=[commit hash]□うこともできます。

また、コミット·ハッシュの□わりにコミット·メッセージの□□を□□することもできます。

```
git commit --squash :/things
```

そこでは、「□」という□□を□いた□□のコミットが□□される。

これらのコミットのメッセージは'fixup!'□まります'fixup!'または'squash!'これらのコミットが□しつぶされる□りのコミットメッセージが□きます。

--autosquashするときには、autosquash / fixup□□を□□するために--autosquashフラグを□□する□□があります。

オンラインでスクワッシュを□む□ https://riptutorial.com/ja/git/topic/598/スクワッシュ

□ 35: ステージング

□□

ステージングは 'ファイル'□□ とはあまり□□ がなく、それぞれのファイル□ の□□ に□□ することはほとんどありません。□□ を□むファイルをステージングし、git は□□ をコミットとして□□ します□ コミットの□□ が□□ のファイルにわたって□われた□□ でも□。

ファイルとコミットの□□ は□□ なように□えるかもしれませんが、この□いを□□ することは、cherry-pick や diff のような□□ な□□ を□□ する□で□□□ です。□ チェリーピックをファイル□□ ツールとして□□ した、□け□れられた□□ の□□ さに□するコメントの□□ を□□してください□。

□□ を□□ するのに□い□□ は□ですか□ それは□□ にありますか□

□なコンセプト□

ファイルは、□□□□ におけるこの2つのより□□□ なメタファーです。ベストプラクティスでは、□□ が□□ されたときにファイル□ が□□ されないように□□ されています□□□ はありません□。

コミットは、ソースコード□□ に□□ のメタファーです。コミットは、バグ□□ のような□□ の□□ に□□ する□□ です。コミットには□□ のファイルが□まれることがよくあります。□□ のマイナーなバグ□□ では、テンプレートと CSS を□□ のファイルに□□ する□□ があります。□□ が□□ され、□□ され、□□□ され、レビューされ、□□ されると、□々のファイルの□□ に□□ を□けて、□□ の□□ として□うことができます。この□□ の□□ ユニットはコミットです。□□ に□□ なのは、レビュー□ のコミットだけに□□ を□てることで、□□ を□けるさまざまなファイルのコード□ を□□ せずに□□ に□□ できることです。

Examples

□□ のファイルをステージングする

コミットするファイルをステージングするには、□のコマンドを□□ します。

```
git add <filename>
```

ファイルに□するすべての□□ のステージング

```
git add -A
```

2.0

```
git add .
```

バージョン2.xでは、 git add .□□ のディレクトリとそのすべてのサブディレクトリにあるファイルに□するすべての□□ がステージングされます。しかし、1.xでは、□□ されたファイルではなく、□しいファイルや□□ されたファイルだけがステージングされます 。

git add -A□□ のバージョンのファイルに□するすべての□□ をステージング git add --all 、 git add -A またはそれに□□ するコマンド git add --all を□□ します。

□□ したファイルをステージングする

```
git rm filename
```

ファイルを git からディスクから□□ せずに□□ するには、-- --cached フラグを□□ します

```
git rm --cached filename
```

□□が□まれているファイルのステ―ジを□□する

```
git reset <filePath>
```

インタラクティブな□□

git add -i □または--interactive □はインタラクティブなインタフェースを□□し、インデックスを□□して、□のコミットに□えたいものを□□することができます。ファイル□□の□□を□□および□□し、□□されないファイルを□□したり、ファイルを□□したりすることができます。また、インデックスに□□する□□のサブセクションを□□したり、□□する□□のチャンクを□□したり、それらのチャンクを□□したり、。 Git□の□くのグラフィカルコミットツール□ git guiなど□にはこのような□□が□まれています。これはコマンドラインバ―ジョンよりも□いやすいかもしれません。

インタラクティブなrebaseの□□で□□しようとしている□□□1□、□々のコミットに□れたい□□ディレクトリの□□を□ませた□□、□きなコミット。

```
$ git add -i
           staged     unstaged path
  1:    unchanged        +4/-4 index.js
  2:        +1/-0      nothing package.json

*** Commands ***
  1: status       2: update       3: revert       4: add untracked
  5: patch        6: diff         7: quit         8: help
What now>
```

この□□の□□□は、インデックスの□□の□□をステ―ジングされた□とステ―ジングされていない□に□□して□しています。

1. index.jsには4□が□□され、4□は□□されています。□□のステ―タスが「□□されていない」と□□されているため、□□はステ―ジングされていません。このファイルがステ―ジングされると、 +4/-4ビットがステ―ジングされた□に□□され、ステ―ジングされていない□には「□もありません」と□□されます。
2. package.jsonには1□が□□され、ステ―ジングされています。ステ―ジングされていない□の□の「□もない」□で□されるようにステ―ジングされているので、これ□□の□□はありません。

□□□はあなたができることを□しています。□□□1-8□または□□□ s、 u、 r、 a、 p、 d、 q、 h □を□□します。

statusは、□の□□の□□と□じ□□が□□されます。

update□□すると、□□の□□を□□してステ―ジングされたコミットをさらに□□できます。

revertは□□□なコミット□□をHEADに□します。

add untrackedを□□すると、□□はバ―ジョンコントロ―ルによって□□されなかったファイルパスを□□することができます。

patchは、さらなる□□のためのstatusと□□に、□□から1つのパスを□□することを□□にする。

diffはコミットするものを□□します。

quitコマンドquit□□します。

help、このコマンドを□□しての□なる□□を□□します。

ハンクによる□□を□える

パッチフラグを□□してコミットのためにステ―ジングされる□□の「ハンクス」を□ることができます。

```
git add -p
```

または

```
git add --patch
```

これにより、diffを□ることができるインタラクティブなプロンプトが□き、それを□めるかどうかを□めることができます。

```
Stage this hunk [y,n,q,a,d,/,s,e,?]?
```

- yの□□□□のために、この□は、コミット
- n□のコミットにこのハンクをステージングしない
- q□□する。この□を□したり、□った□□をしたりしないでください
- □□この□とファイル□のすべての□にハンク
- dこのファイルを□で□□しないでください
- g□きたい□を□んでください
- /□えられた□□□□にマッチする□を□□する
- jはこの□を□□にしておき、□の□□□□を□てください
- Jはこの□を□□にしておき、□の□を□てください
- kこの□を□□にしておく、□の□□□□を□□
- Kはこの□を□□にしておきます、□の□を□てください
- sが□さい□□に□□の□を□□しました
- □□□□で□□の□を□□
- □□□□ヘルプ

これにより、コミットしたくない□□を□□にキャッチできます。

git add --interactiveとpを□□することでこれを□くこともできます。

□□□な□□を□□

コミットのためにステージングされたハンズクを□□するには□

```
git diff --cached
```

オンラインでステージングを□む□ https://riptutorial.com/ja/git/topic/244/ステージング

第 36 章: バイゼクト/□□ったコミットの□□

□□

- git bisect <subcommand> <options>
- git bisect start <bad> [<good>...]
- git bisect reset
- git bisect good
- git bisect bad

Examples

バイナリ□□□**git bisect**□

git bisectは、バイナリ□□を□ってどのコミットがバグを□□したのかを□つけることができます。

2つのコミット□□を□□することによってセッションを□□□することから□めます□バグの□に□いコミット、バグの□に□いコミットです。□□□に、□いコミットはHEADです。

```
# start the git bisect session
$ git bisect start

# give a commit where the bug doesn't exist
$ git bisect good 49c747d

# give a commit where the bug exist
$ git bisect bad HEAD
```

gitはバイナリ□□を□□します□リビジョンを□□に□□し、リポジトリを□□リビジョンに□り□えます。リビジョンが□いか□いかを□□するためにコードを□べます。

```
# tell git the revision is good,
# which means it doesn't contain the bug
$ git bisect good

# if the revision contains the bug,
# then tell git it's bad
$ git bisect bad
```

gitはあなたの□□に□じて□いリビジョンの□りのサブセットごとにバイナリ□□を□けます。 gitは、あなたのフラグが□□っていない□り、バグが□□されたリビジョンを□□に□□する、□□のリビジョンを□□します。

その□、 git bisect resetを□□してbisectセッションを□□し、HEADに□ります。

```
$ git bisect reset
```

バグをチェックできるスクリプトがあれば、□の□□でプロセスを□□□できます。

```
$ git bisect run [script] [arguments]
```

ここで[script]は[script]へのパスで、 [arguments]はスクリプトに□すべき□□です。

このコマンドを□□するとバイナリ□□が□□□に□□され、スクリプトの□□コードに□じて□ステップでgit bisect goodまたはgit

bisect bad□□されます。 0で□□することを□してgood 1-124、126、または127で□□する□い□しています。 125はスクリプトがそのリビジョンをテストできないことを□します□ git bisect skipを□□します□。

□□□で□□ったコミットを□つける

あなたが□にある□□masterブランチと□かが□□□が□□されました□□□□りに□□していないが、あなたがどこかわかりません。あなたが□っていることは、□□のリリース□タグ□けされているか、コミットハッシュを□っていて、ここでold-rel□ることができる□で□□していたことだけです。

Gitは、□□に□ないステップ□□2□□□□で□□を□□した□□ったコミットを□つけ□し、□けを□ています。

まず□□にバイセクションを□□します。

```
git bisect start master old-rel
```

これはmasterが□れたリビジョン□または□□の□れたバージョン□であり、 old-relが□□の□□のバージョンであることをgitに□えます。

Gitは□□のコミットの□□で□り□されたヘッドをチェックします。□、あなたはあなたのテストを□うことができます。それが□□するかどうかに□じて

```
git bisect good
```

または

```
git bisect bad
```

。このコミットをテストすることができない□□は、□□にgit reset□い、それをテストすることができます。git willlがこれを□□します。

いくつかのステップの□でgitは□□したコミットハッシュを□□します。

バイセクトプロセスを□□するには、□に□□します

```
git bisect reset
```

gitは□□の□□を□□します。

オンラインでバイゼクト/□□ったコミットの□□を□む□ https://riptutorial.com/ja/git/topic/3645/バイゼクト-□□ったコミットの□□

□ 37: バンドル

□□

この□□を□うための□は、リポジトリ□□の□□から□まるバンドルを□□することから□めることです。

```
git bundle create initial.bundle master
git tag -f some_previous_tag master  # so the whole repo does not have to go each time
```

その□□のバンドルをリモートマシンに□□します。そして

```
git clone -b master initial.bundle remote_repo_name
```

Examples

ローカルマシンで**git**バンドルを□□し、それを□のマシンで□□する

□□によっては、ネットワーク□□のないマシンでgitリポジトリのバージョンを□□することができます。バンドルを□□すると、あるマシンのリポジトリにあるgitオブジェクトと□□をパッケージ□し、□のマシンのリポジトリにインポートすることができます。

```
git tag 2016_07_24
git bundle create changes_between_tags.bundle [some_previous_tag]..2016_07_24
```

□らかの□で**changes_between_tags.bundle**ファイルをリモートマシンに□□します。□えば、□□ドライブを□して。□□それを□っていれば□

```
git bundle verify changes_between_tags.bundle  # make sure bundle arrived intact
git checkout [some branch]        # in the repo on the remote machine
git bundle list-heads changes_between_tags.bundle # list the references in the bundle
git pull changes_between_tags.bundle [reference from the bundle, e.g. last field from the
previous output]
```

□も□□です。リモートリポジトリに□□を□えたら、デルタをバンドルすることができます。□えばサムドライブなどの□□をローカルリポジトリにマージして、マシン□でgit 、 ssh 、 rsync 、またはhttpプロトコルの□□アクセスを□□とせずに□□をとることができます。

オンラインでバンドルを□む□ https://riptutorial.com/ja/git/topic/3612/バンドル

□ 38: ファイルとフォルダを□□する

□□き

このトピックでは、□□なファイル□またはファイルの□□□をGitリポジトリに□□しないようにする□□について□□します。いくつかの□□□グローバルまたはローカルの.gitignore、 .git/exclude、 git update-index --assume-unchanged、およびgit update-index --skip-tree □がありますが、Gitはコンテンツを□□しています。□□すると、□□にはフォルダの□□ □ファイルなど□は□□されます。とにかく□□できないので、□のフォルダはデフォルトでは□□されます。

Examples

.gitignoreファイルでファイルとディレクトリを□□する

あなたのリポジトリに1つ□□の.gitignoreファイルを□□することで、Gitが□□のファイルとディレクトリを□□するようにすることができます。

ソフトウェアプロジェクトでは、 .gitignoreは□□、ビルドプロセス□または□□□に□□されるファイルやディレクトリのリストが□まれています。 .gitignoreファイルのエントリには、 .gitignore□す□□またはパスが□まれます。

1. キャッシュ、ログファイル、コンパイル□みコードなどの□□□なリソース
2. □の□□□と□□してはならないローカル□□ファイル
3. ログインパスワード、□、□□□□などの□□□□を□むファイル

□□□ディレクトリに□□すると、ルールはリポジトリ□□のすべてのファイルとサブディレクトリに□□□に□□されます。サブディレクトリに□□すると、その□□のディレクトリとそのサブディレクトリにルールが□□されます。

ファイルまたはディレクトリを□□すると、□のようになりません。

1. Gitによって□□される
2. git statusやgit diffなどのコマンドによって□□される
3. git add -Aようなコマンドでステージングされます

トラッキングされたファイルを□□する□□がある□□な□□は、□□な□□が□□です。 See□ Gitリポジトリにすでにコミットされているファイルを□□する 。

□

globファイルパターンに□づいて、 .gitignoreファイル□のルールの□□□な□をいくつか□します 。

```
# Lines starting with `#` are comments.

# Ignore files called 'file.ext'
file.ext

# Comments can't be on the same line as rules!
# The following line ignores files called 'file.ext # not a comment'
file.ext # not a comment

# Ignoring files with full path.
# This matches files in the root directory and subdirectories too.
# i.e. otherfile.ext will be ignored anywhere on the tree.
dir/otherdir/file.ext
otherfile.ext

# Ignoring directories
```

```
# Both the directory itself and its contents will be ignored.
bin/
gen/

# Glob pattern can also be used here to ignore paths with certain characters.
# For example, the below rule will match both build/ and Build/
[bB]uild/

# Without the trailing slash, the rule will match a file and/or
# a directory, so the following would ignore both a file named `gen`
# and a directory named `gen`, as well as any contents of that directory
bin
gen

# Ignoring files by extension
# All files with these extensions will be ignored in
# this directory and all its sub-directories.
*.apk
*.class

# It's possible to combine both forms to ignore files with certain
# extensions in certain directories. The following rules would be
# redundant with generic rules defined above.
java/*.apk
gen/*.class

# To ignore files only at the top level directory, but not in its
# subdirectories, prefix the rule with a `/`
/*.apk
/*.class

# To ignore any directories named DirectoryA
# in any depth use ** before DirectoryA
# Do not forget the last /,
# Otherwise it will ignore all files named DirectoryA, rather than directories
**/DirectoryA/
# This would ignore
# DirectoryA/
# DirectoryB/DirectoryA/
# DirectoryC/DirectoryB/DirectoryA/
# It would not ignore a file named DirectoryA, at any level

# To ignore any directory named DirectoryB within a
# directory named DirectoryA with any number of
# directories in between, use ** between the directories
DirectoryA/**/DirectoryB/
# This would ignore
# DirectoryA/DirectoryB/
# DirectoryA/DirectoryQ/DirectoryB/
# DirectoryA/DirectoryQ/DirectoryW/DirectoryB/

# To ignore a set of files, wildcards can be used, as can be seen above.
# A sole '*' will ignore everything in your folder, including your .gitignore file.
# To exclude specific files when using wildcards, negate them.
# So they are excluded from the ignore list:
!.gitignore

# Use the backslash as escape character to ignore files with a hash (#)
# (supported since 1.6.2.1)
\#*#
```

ほとんどの.gitignoreファイルは、さまざまな□□で□□□なファイルなので、ここでは、プロジェクトに□□またはコピー/□□するための□□でリストされた.gitignoreサンプルファイルのセットを.gitignoreします。また、□しいプロジェクトでは、 オンラインツールを□□してスター□ファイルを□□□□することを□□することもできます 。

.gitignoreの□の□□

.gitignoreファイルは、リポジトリの□□としてコミットされることを□□しています。□□ルールをコミットせずに□□のファイルを□□する□□は、□のオプションがあります。

- .git/info/excludeファイルを□□します□ .gitignoreと□じ□□を□□します□。ルールはリポジトリの□□□でグロ□バルになります。
- すべてのロ□カルリポジトリに□□ルールを□□するグロ□バルなgitignoreファイルを□□します□

さらに、グロ□バルなgit□□を□□せずに、□□されたファイルのロ□カル□□を□□することができます□

- git update-index --skip-worktree [<file>...] □マイナ□ロ□カル□□
- git update-index --assume-unchanged [<file>...] □プロダクションの□□ができていて□□のないファイル

さらなるオプションについては、□□のフラグとgit update-indexドキュメントの □□□の□□ を□□してください 。

□□されたファイルのクリーンアップ

git clean -Xを□□して、□□されたファイルをクリーンアップすることができます□

```
git clean -Xn #display a list of ignored files
git clean -Xf #remove the previously displayed files
```

□□ -X □□□□□ は□□されたファイルのみを□□します。□□されていないファイルも□□するには、 -x □□□□なし□ を□□します。

□□については、 git cleanドキュメントを□□してください。

□□については、Gitのマニュアルを□□してください。

.gitignoreファイルの□□

パターンを□□してファイルを□□しますが、□□がある□□は、□□に□□□□□□ を□けます。□えば□

```
*.txt
!important.txt
```

□の□は、 important.txtという□□のファイルを□いて、□□□が.txtファイルをすべて□□するようGitに□□します。

ファイルが□□されたフォルダにある□□は、そのファイルを□□に□み□むことはできません 。

```
folder/
!folder/*.txt
```

この□では、フォルダ□のすべての.txtファイルは□□されたままです。

□しい□□は、フォルダ□□を□の□に□れなおしてから*でfolderすべてのファイルを□□し、□□に*.txtを□のようにfolderに□*.txtます。

```
!folder/
folder/*
!folder/*.txt
```

□ □□□□で□まるファイル□の□□は、□□□を2つ□□するか、 \エスケープしてください□

```
!!includethis
\!excludethis
```

グローバル.gitignoreファイル

Gitがすべてのリポジトリで□□のファイルを□□するようにするには、□□またはコマンドプロンプトで□のコマンドを□□してグローバルな.gitignoreを□□します。

```
$ git config --global core.excludesfile <Path_To_Global_gitignore_file>
```

これでGitは□リポジトリの.gitignoreファイルに□えてこれを□□します。これに□するルールは□のとおりです。

- グローバルな.gitignoreファイルを□□しているときに、ローカルの.gitignoreファイルにファイルが□□□に□まれている□□は、ローカルの.gitignoreが□□されます□ファイルがインクルードされます□
- リポジトリは□□のマシンに□□されている□□は、グローバル.gigignoreグローバルでのPC□、□□されたファイルは、レポに□し□げられるように、すべてのマシンにロードしなければならないか、または□なくともそれを□める.gitignoreそれを□□しません。これは、プロジェクトがチームによって□□されている□□、リポジトリ□□の.gitignoreはグローバルなものより□いアイデアです

このファイルには、□□の□□、□えばOSXプラットフォーム、マシンまたはユーザーを□□するには□い□□です.DS_Store、WindowsのThumbs.dbやVimが*.ext~と*.ext.swpあなたがリポジトリにそれらを□□しない□□は□□します。だから、_MACOSX□く1□のチームメンバーは、すべての.DS_STOREと_MACOSX □□□には□に□たない□を□□することができますが、Windows□の□のチームメンバーはすべてのthumbs.bd

すでにGitリポジトリにコミットされているファイルは□□する

□にGitリポジトリにファイルを□□していて、それを□□しないようにして□□□のコミットには□□しないように□、それをインデックスから□□することができます□

```
git rm --cached <file>
```

これにより、リポジトリからファイルが□□され、それ□□の□□がGitによって□□されなくなります。 --cachedオプションは、ファイルが□□□に□□されていないことを□□します。

□□に□□されたファイルの□□は、Gitの□□から□ることができます。

□□からファイルを□□した□に□の□かがリポジトリから□き□すと、 そのコピーは□□□に□□されることに□□してください 。

Gitは、ファイルの□□ディレクトリのバージョンが□□のものであるようにして、□わりに "skip worktree "ビットを□ってインデックスバージョンを□み□むことができます□そのため、□□は□□されます□。

```
git update-index --skip-worktree <file>
```

□き□みはこのビットの□□を□けず、コンテンツの□□□が□□□です。□□な□□された□□を□うことはありません。□□、このビットはstashingと□□します。このビットを□□するには、

```
git update-index --no-skip-worktree <file>
```

Git□に□をつけ、ファイルを□べずにファイルが□□されていないと□□することは□□っていることがあります。□□すると、ファイルからのそれ□□の□□を□□して、インデックスから□□することなく□えます□

```
git update-index --assume-unchanged <file>
```

これにより、gitはファイル□の□□を□□します□このファイルに□□を□えたり□したりすると□□される□□が□われることに□□してください □

gitにこのファイルを□□□にさせたい□□は、□のコマンドを□□してください□

```
git update-index --no-assume-unchanged <file>
```

ファイルが□□されているかどうかを□□する

`git check-ignore`コマンドは、Gitによって□□されたファイルを□□します。

コマンドラインでファイル□を□することができ、 `git check-ignore`は□□されるファイル□をリストします。□えば□

```
$ cat .gitignore
*.o
$ git check-ignore example.o Readme.md
example.o
```

ここでは、*。oファイルのみが.gitignoreで□□されているので、Readme.mdは`git check-ignore`□□には□□されません。

.gitignoreがファイルを□□する□を□□したい□□は、-vを`git check-ignore`コマンドに□□します。

```
$ git check-ignore -v example.o Readme.md
.gitignore:1:*.o        example.o
```

Git 1.7.6□□では、□□されたファイルを□□するために`git status --ignored`を□□することもできます。これに□する□□は、□□□□または`.gitignore`によって□□されるファイルの□□を□□してください 。

サブフォルダ□のファイルを□□する□□□の**gitignore**ファイル□

□のようなリポジトリ□□があるとします。

```
examples/
    output.log
src/
    <files not shown>
    output.log
README.md
```

`examples`ディレクトリの`output.log`は□□であり、プロジェクトの□□を□るためには□□ですが、 `src/`□にあるものはデバッグ□に□□され、□□やリポジトリの□□には□□しないようにしてください。

このファイルを□□するには、2つの□□があります。 `.gitignore`ファイルに□□ディレクトリのルートに□□パスを□れることができます□

```
# /.gitignore
src/output.log
```

□わりに、 `src/`ディレクトリに`.gitignore`ファイルを□□し、これに□□するファイルを□□することもできます`.gitignore` □

```
# /src/.gitignore
output.log
```

□□のディレクトリ□のファイルを□□する

どのディレクトリでも `foo.txt`というファイルを□□するには、その□□を□きます□

```
foo.txt # matches all files 'foo.txt' in any directory
```

ツリーの□□のみでファイルを□□する□□は、 `** pattern`を□□して□□のディレクトリのサブディレクトリを□□することができま

す。

```
bar/**/foo.txt # matches all files 'foo.txt' in 'bar' and all subdirectories
```

または、bar/ディレクトリに.gitignoreファイルを□□することもできます。□の□に□□するのは、ファイルbar/.gitignoreを□の□□で□□することです。

```
foo.txt # matches all files 'foo.txt' in any directory under bar/
```

ルールを□□することなくローカルでファイルを□□する

.gitignoreはファイルをローカルでは□□しますが、リポジトリにコミットされ、□のコントリビュータやユーザと□□されることを□□しています。グローバル.gitignoreを□□することはできますが、すべてのリポジトリはそれらの□□を□□します。

リポジトリ□の□□のファイルをローカルで□□し、そのファイルをリポジトリの□□にしない□□は、リポジトリ□の.git/info/exclude□□します。

□えば□

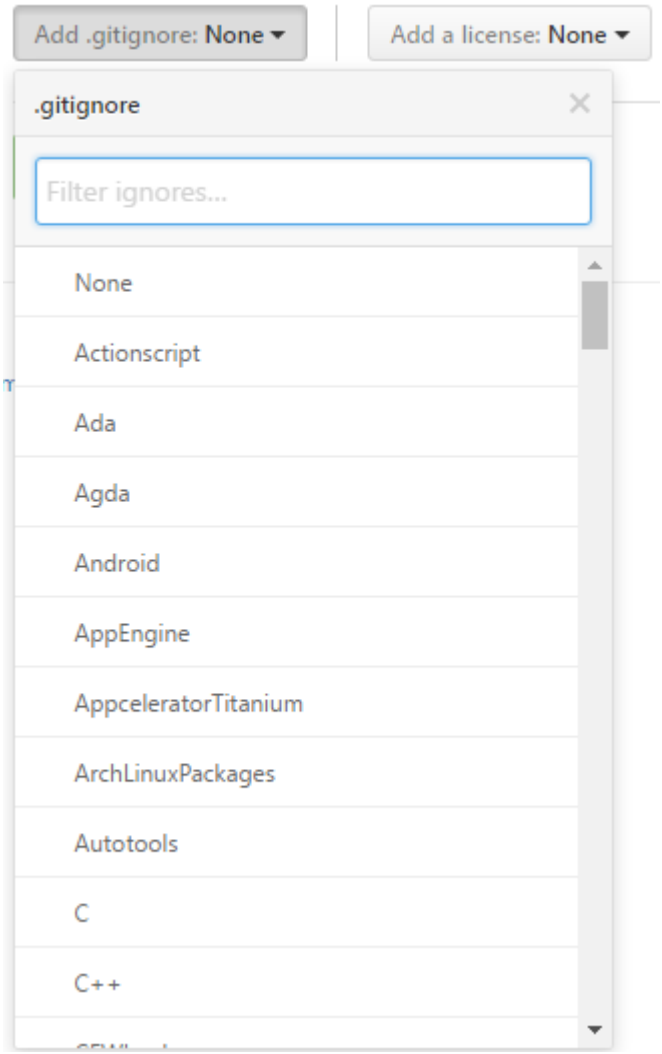
```
# these files are only ignored on this repo
# these rules are not shared with anyone
# as they are personal
gtk_tests.py
gui/gtk/tests/*
localhost
pushReports.py
server/
```

□□の.gitignoreテンプレート

.gitignoreファイルにどのルールをリストするかわからない□□や、□□□に□け□れられている□□をプロジェクトに□□したい□□は、.gitignoreファイルを□□または□□できます□

- https://www.gitignore.io/
- https://github.com/github/gitignore

GitHubやBitBucketなどの□くのホスティングサービスは、□□しているプログラミング□□やIDEに□づいて.gitignoreファイルを□□する□□を□□します。



ファイルへの□□の□□を□□する□□□せずに□

Gitでファイルを□□したいが、その□の□□を□□することがあります。

update-indexを□ってファイルやディレクトリの□□を□□するようにGitに□□する□

```
git update-index --assume-unchanged my-file.txt
```

□□のコマンドは、Gitにmy-file.txtが□□されていないと□□しmy-file.txt□□を□□したり□□したりしないように□□します。
ファイルは□□としてリポジトリに□□します。

これは、デフォルトを□□したり、ローカル□□のオーバーライドを□□するのに□□です。□□

```
# create a file with some values in
cat <<EOF
MYSQL_USER=app
MYSQL_PASSWORD=FIXME_SECRET_PASSWORD
EOF > .env

# commit to Git
git add .env
git commit -m "Adding .env template"

# ignore future changes to .env
git update-index --assume-unchanged .env
```

```
# update your password
vi .env

# no changes!
git status
```

ファイルの□□だけを□□する**[stub]**

□□によっては、コミットまたは□□したくないファイルにローカルで□□を□えたい□□があります。□□□には、ローカル□□は.gitignoreに□くことができる□のファイルに□□する□□がありますが、□には□□□な□□□として、チェックインされたファイルにローカルなものを□たせることが□に□ちます。

Gitは、クリーンなフィルタを□ってそれらの□を"□せ"ることができます。□らはdiffで□□されません。

ファイルfile1.cスニペットを□のようにします。

```
struct settings s;
s.host = "localhost";
s.port = 5653;
s.auth = 1;
s.port = 15653; // NOCOMMIT
s.debug = 1; // NOCOMMIT
s.auth = 0; // NOCOMMIT
```

NOCOMMIT□をどこにでも□□する□□はありません。

.git/configようなGit□□ファイルにこれを□□して"nocommit"フィルタを□□します□

```
[filter "nocommit"]
    clean=grep -v NOCOMMIT
```

これを.git/info/attributesまたは.gitmodules□□□または□□□し.git/info/attributes □

```
file1.c filter=nocommit
```

あなたのNOCOMMIT□はGitから□されています。

□□□

- クリーンフィルタを□□すると、□にWindowsではファイルの□□が□くなります。
- Gitが□□すると、□□された□がファイルから□えることがあります。それは□れフィルタで□ち□すことができますが、それはよりトリッキーです。
- Windowsではテストされていない

□□されたファイルの□□を□□する。[スタブ]

.gitignoreと.git/info/excludeは、□□されていないファイルに□してのみ□□します。

□□されたファイルに□してignoreフラグを□□するには、update-indexコマンドを□□します。

```
git update-index --skip-worktree myfile.c
```

これを□に□すには、□□を□□します。

```
git update-index --no-skip-worktree myfile.c
```

この□□をグローバルgit□□に□□して、より□□なgit hide、git unhide、git hiddenコマンドを□□することができます□

```
[alias]
    hide   = update-index --skip-worktree
    unhide = update-index --no-skip-worktree
    hidden  = "!git ls-files -v | grep ^[hsS] | cut -c 3-"
```

また、update-index□□で--assume-unchangedオプションを□□することもできます

```
git update-index --assume-unchanged <file>
```

□□のためにこのファイルをもう□□□たい□□は、

```
git update-index --no-assume-unchanged <file>
```

--assume-unchangedフラグが□□されている□□、ユーザはファイルを□□しないことを□□し、□□ツリーファイルがインデックスに□□されているものと□□するとGitに□□させます。インデックスでこのファイルを□□する□□がある□□□えば、コミットでマージするとき。したがって、□□□のファイルがアップストリームで□□された□□は、□□で□□を□□する□□があります。この□□は、パフォーマンスに□□が□かれます。

--skip-worktreeフラグは、ファイルがローカルで□□され、あなたが□□って□□をコミットしたくないために□□のファイルに□れないように□□するときに□□です□つまり、□□のファイルに□して□□された□□/プロパティファイル□□□。 Skip-worktreeは、□□とも□□されている□□は、□□されていません。

□にコミットされたファイルを□□するが、.gitignoreにインクルードする

□□によってはファイルがgitによって□□されていることがありますが、□でgitignoreに□□を□□するために□□されました。.gitignoreに□□する□に、そのようなファイルをクリーンアップするのを□れるのは、□□に□□□なシナリオです。この□□、□いファイルは□□としてリポジトリにふら□がっています。

この□□を□□するには、リポジトリ□のすべてを「ドライラン」で□□してから、すべてのファイルを□に□すことができます。□□□の□□がなく、-- --cachedパラメータが□されている□り、このコマンドは□□するのがかなり□□です□

```
# Remove everything from the index (the files will stay in the file system)
$ git rm -r --cached .

# Re-add everything (they'll be added in the current state, changes included)
$ git add .

# Commit, if anything changed. You should see only deletions
$ git commit -m 'Remove all files that are in the .gitignore'

# Update the remote
$ git push origin master
```

□のフォルダを□□する

Gitが□のフォルダを□□してコミットすることはできません.Gitがファイルを□□し、そのディレクトリをそれらに□□するからです。コミットが□り□□が□□します。これを□□するには、2つの□□があります。

□□1□ .gitkeep

これを□□するには、Gitのために.gitkeepファイルを□□してフォルダを□□する□□があります。これを□うには、□□なディレクトリを□□し、フォルダに.gitkeepファイルを□□する.gitkeepです。このファイルは□□で、フォルダを□□する□□の□□はありません。 Windowsでこれを□うには□ファイル□□□□が□□です□、ディレクトリにgit bashをオープンし、□のコマンドを□□してください□

```
$ touch .gitkeep
```

このコマンドは、□□のディレクトリに□の.gitkeepファイルを□□するだけです

□□2□ dummy.txt

もう1つのハックは□□と□□によく□ていますが、□じ□□を.gitkeepことができますが、 dummy.txt□わりにdummy.txt□わりに□□してdummy.txt。 コンテキストメニューを□□してWindowsで□□に□□できるという□□があります。また、□□いメッセージを□することもできます。また、 .gitkeepファイルを□□して□のディレクトリを□□することもできます。 .gitkeepは□のダイレクトィを□□するために□□される□のファイルです。

.gitignoreによって□□されるファイルを□つける

gitで□□されるすべてのファイルを□□のディレクトリにコマンドで□□することができます□

```
git status --ignored
```

だから、このようなリポジトリ□□があれば□

```
.git
.gitignore
./example_1
./dir/example_2
./example_2
```

...と.gitignoreファイルが□まれています□

```
example_2
```

...コマンドの□□よりも□

```
$ git status --ignored

On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

.gitignore
.example_1

Ignored files:
  (use "git add -f <file>..." to include in what will be committed)

dir/
example_2
```

ディレクトリ□で□□□に□□されるファイルを□□□□するには、□□のパラメータを□□する□□があります--untracked-files=all

□□は□のようになります。

```
$ git status --ignored --untracked-files=all
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

.gitignore
example_1
```

```
Ignored files:
  (use "git add -f <file>..." to include in what will be committed)

dir/example_2
example_2
```

オンラインでファイルとフォルダを□□するを□む□ https://riptutorial.com/ja/git/topic/245/ファイルとフォルダを□□する

□ 39: フィルタブランチで□□を□き□す

Examples

コミットの□□を□□する

□□フィルタを□□して、コミットの□□□を□□することができます。スクリプト□の$GIT_AUTHOR_NAME□□してエクスポートして、□がコミットを□□したかを□□してください。

□のような□□のファイルfilter.shを□□します□

```
if [ "$GIT_AUTHOR_NAME" = "Author to Change From" ]
then
    export GIT_AUTHOR_NAME="Author to Change To"
    export GIT_AUTHOR_EMAIL="email.to.change.to@example.com"
fi
```

□に、コマンドラインからfilter-branchを□□しfilter-branch。

```
chmod +x ./filter.sh
git filter-branch --env-filter ./filter.sh
```

gitコミッター—をコミットの□□に□□する

コミット□□commit1..commit2□えられたこのコマンドは、git commit authorもgit committerになるように□□を□き□えます。

```
git filter-branch -f --commit-filter \
   'export GIT_COMMITTER_NAME=\"$GIT_AUTHOR_NAME\";
    export GIT_COMMITTER_EMAIL=\"$GIT_AUTHOR_EMAIL\";
    export GIT_COMMITTER_DATE=\"$GIT_AUTHOR_DATE\";
    git commit-tree $@' \
    -- commit1..commit2
```

オンラインでフィルタブランチで□□を□き□すを□む□ https://riptutorial.com/ja/git/topic/2825/フィルタブランチで□□を□き□す

□ 40: フック

□□

- .git / hooks / applypatch-msg
- .git / hooks / commit-msg
- .git / hooks / post-update
- .git / hooks / pre-applypatch
- .git / hooks / pre-commit
- .git / hooks / prepare-commit-msg
- .git / hooks / pre-push
- .git / hooks / pre-rebase
- .git / hooks / update

□□

--no-verifyまたは-n□□されたgitコマンドのすべてのロ―カルフックをスキップします。
□□ git commit -n

このペ―ジの□□はGitの□□ドキュメントとアトラシアンから□められまし た。

Examples

Commit-msg

このフックはprepare-commit-msgフックに□ていますが、ユ―ザ―が□□よりもコミットメッセ―ジを□□した□に□び□されます。これは□□、コミットメッセ―ジの□□が□□っている□□に□□□に□□するために□□されます。

このフックに□される□□の□□は、メッセ―ジを□むファイルの□□です。ユ―ザ―が□□したメッセ―ジが□に□らない□□は、このファイルをインプレ―ス□ prepare-commit-msgと□じ□に□□するか、ゼロ□□のステ―タスで□□することによってコミットを□□に□□できます。

□の□は、コミット·メッセ―ジにワ―ド□□の□に□□があるかどうかをチェックするために□□されます

```
word="ticket [0-9]"
isPresent=$(grep -Eoh "$word" $1)

if [[ -z $isPresent ]]
  then echo "Commit message KO, $word is missing"; exit 1;
  else echo "Commit message OK"; exit 0;
fi
```

ロ―カルフック

ロ―カルフックは、ロ―カルのリポジトリにのみ□□します。□□□□はロ―カルフックを□□することができるため、コミットポリシ―を□□する□□として□□に□□することはできません。□□□は□□のガイドラインを□□し、□□□な□□を□□することが□□になるように□□されています。

ロ―カルコミットには、□□コミット、□□コミットメッセ―ジ、コミットメッセ―ジ、ポストコミット、ポストチェックアウト、およびプレリベ―スがあります。

□□の4つのフックはコミットに□□しており、コミットのライフサイクルで□パートを□らかの□で□□することができます。□□の2つは、git checkoutコマンドとgit rebaseコマンドのいくつかの□□アクションまたは□□□チェックを□□できるようにします。

すべての"プリ"フックでは、□□しようとしている□□を□□できますが、 "ポスト"フックは□に□□に□□されます。

ポストチェックアウト

このフックはpost-commitフックと□□に□□しますが、 git checkout□□をチェックアウトすると□び□されます。これは、□□の□□になる□□□□ファイルの□□ディレクトリをクリアするのに□□なツ—ルです。

このフックは3つのパラメ—タを□け□ります□

1. □のHEADのref、
2. □しいHEADのref、
3. ブランチ·チェックアウトかファイル·チェックアウトかを□すフラグ□それぞれ1または0 □。

その□□ステ—タスは、 git checkoutコマンドに□□しません。

コミット□

このフックは、 commit-msgフックの□□に□び□されます。 git commit□□の□□を□□することはできないため、□に□□□□で□□されます。

スクリプトはパラメ—タを□らず、□□ステータスは□してコミットに□□しません。

□□□

このフックは、□□したプッシュ□□の□に□び□されます。□□、□□の□□で□□されます。

スクリプトにはパラメ—タはありませんが、□□□□からのpre-receiveと□じ□□が□□されます。

```
<old-value> <new-value> <ref-name>
```

□□コミット

このフックはgit commitを□□するたびに□□され、 git commitしようとしていることを□□します。このフックを□□して、コミットしようとしているスナップショットを□□できます。

このタイプのフックは、□□テストを□□して、□□コミットによってプロジェクトの□□の□□が□□されないことを□□するのに□□です。このタイプのフックは、□□またはEOLエラ—をチェックすることもできます。

□□はプリコミットスクリプトに□されず、ゼロ□□の□□で□□するとコミット□□が□□されます。

Prepare-commit-msg

このフックは、テキストエディタにコミットメッセ—ジを□□するために、 pre-commitフックの□に□び□されます。これは、□□、□□またはマ—ジされたコミットのために□□□に□□されたコミットメッセ—ジを□□するために□□されます。

このフックには1□3つの□□が□されます。

- メッセ—ジを□む□□ファイルの□□。
- コミットのタイプ。
    - メッセ—ジ□ -mまたは-Fオプション□、
    - テンプレート□ -tオプション□、
    - マ—ジ□マ—ジコミットの□□□、または
    - スカッシュ□□のコミットが□れている□□□
- □□するコミットのSHA1ハッシュ。これは、 -c 、 -C 、 ---amendオプションが□えられた□□にのみ□えられます。

pre-commitと□□に、ゼロ□□の□□で□□すると、コミットが□□されます。

プレ·リベ—ス

このフックは、 git rebaseがコード□□を□□する□に□び□されます。このフックは、□□、リベ―ス□□が□□であることを□□するために□□されます。

このフックには2つのパラメ―タがあります□

1. シリ―ズがフォ―クされた□□ブランチ、および
2. ブランチはリベ―スされます□□□のブランチをリベ―スするときは□です□。

ゼロ□□のステ―タスで□□することによって、リベ―ス□□を□□することができます。

□□□□

このフックは□かがgit pushを□ってコミットをリポジトリにプッシュするたびに□□されます。□にプッシュの□□であるリモ―トリポジトリに□□し、□の□ロ―カル□リポジトリには□□しません。

□□が□□される□にフックが□□されます。これは、□□、あらゆる□□の□□ポリシ―を□□するために□□されます。

スクリプトにはパラメ―タはありませんが、プッシュされている□refは、□□□□の□の□にあるスクリプトに□の□□で□されます。

```
<old-value> <new-value> <ref-name>
```

□□

このフックはpre-receive□に□び□され、□じ□□で□□します。□かが□□に□□される□に□び□されますが、□□にすべてのrefではなく、pushされたrefごとに□々に□び□されます。

このフックは、□の3つの□□を□け□ります。

- □□される□□の□□、
- refに□□されている□いオブジェクト□
- refに□□されている□しいオブジェクト□。

これはpre-receive□された□じ□□ですが、 updateがrefごとに□□に□び□されるため、いくつかのrefを□□して□のものを□□することができます。

プッシュプッシュ

Git 1.8.2□□で□□□□です。

1.8

プッシュプッシュフックを□□して、プッシュの□□を□ぐことができます。これは、□□のブランチへの□った□□プッシュのブロック、または□□されたチェックが□□した□□のプッシュのブロック□ユニットテスト、□□□を□む。

□□にプッシュフックは□に□□のファイル□□することによって□□されるpre-push□.git/hooks/ □ファイルが□□□□である、と□□とし□□□を □□□しchmod +x ./git/hooks/pre-push。

Hannah Wolfeの□□を□□します。

```
#!/bin/bash

protected_branch='master'
current_branch=$(git symbolic-ref HEAD | sed -e 's,.*/\(.*\),\1,')

if [ $protected_branch = $current_branch ]
then
    read -p "You're about to push master, is that what you intended? [y|n] " -n 1 -r <
/dev/tty
```

```
    echo
    if echo $REPLY | grep -E '^[Yy]$' > /dev/null
    then
        exit 0 # push will execute
    fi
    exit 1 # push will not execute
else
    exit 0 # push will execute
fi
```

Volkan Unsalの□では、プッシュを□□する□にRSpecテストが□□することを□□しています□

```
#!/usr/bin/env ruby
require 'pty'
html_path = "rspec_results.html"
begin
  PTY.spawn( "rspec spec --format h > rspec_results.html" ) do |stdin, stdout, pid|
  begin
    stdin.each { |line| print line }
  rescue Errno::EIO
  end
end
rescue PTY::ChildExited
  puts "Child process exit!"
end

# find out if there were any errors
html = open(html_path).read
examples = html.match(/(\d+) examples/)[0].to_i rescue 0
errors = html.match(/(\d+) errors/)[0].to_i rescue 0
if errors == 0 then
  errors = html.match(/(\d+) failure/)[0].to_i rescue 0
end
pending = html.match(/(\d+) pending/)[0].to_i rescue 0

if errors.zero?
  puts "0 failed! #{examples} run, #{pending} pending"
  # HTML Output when tests ran successfully:
  # puts "View spec results at #{File.expand_path(html_path)}"
  sleep 1
  exit 0
else
  puts "\aCOMMIT FAILED!!"
  puts "View your rspec results at #{File.expand_path(html_path)}"
  puts
  puts "#{errors} failed! #{examples} run, #{pending} pending"
  # Open HTML Ooutput when tests failed
  # `open #{html_path}`
  exit 1
end
```

あなたが□ることができるように、□くの□□□がありますが、□いことが□こった□□はexit 0をexit 1、□いことが□きた□□はexit 1をexit 1ます。いつでもexit 1をexit 1するとプッシュは□□され、コードはgit push...□□する□の□□になりgit push...。

クライアントサイドフックを□□する□□は、プッシュで「--no-verify」オプションを□□することで、すべてのクライアントサイドフックをスキップできます。プロセスを□□するためにフックに□っているのであれば、□え□きる□□□があります。

ドキュメント□ https : //git-scm.com/docs/githooks#_pre_push
□□サンプル□ https : //github.com/git/git/blob/87c86dd14abe8db7d00b0df5661ef8cf147a72a3/templates/hooks--pre-push.sample

コミットする□にMavenビルド□ または□のビルドシ ステム□ を□□する

.git/hooks/pre-commit

```
#!/bin/sh
if [ -s pom.xml ]; then
    echo "Running mvn verify"
    mvn clean verify
    if [ $? -ne 0 ]; then
        echo "Maven build failed"
        exit 1
    fi
fi
```

□□のプッシュを□のリポジトリに□□□に□□する

post-receiveフックを□□して、□ってくるプッシュを□□□に□のリポジトリに□□することができます。

```
$ cat .git/hooks/post-receive

#!/bin/bash

IFS=' '
while read local_ref local_sha remote_ref remote_sha
do

  echo "$remote_ref" | egrep '^refs\/heads\/[A-Z]+-[0-9]+$' >/dev/null && {
    ref=`echo $remote_ref | sed -e 's/^refs\/heads\///'`
    echo Forwarding feature branch to other repository: $ref
    git push -q --force other_repos $ref
  }

done
```

この□では、 egrep regexpは□□のブランチ□□を□ します□ ここでは、JIRAの□□に□□ を□けるためにJIRA-12345が□□されています□。もちろん、すべてのブランチを□□したい□□は、この□□をオフにすることができます。

オンラインでフックを□む□ https://riptutorial.com/ja/git/topic/1330/フック

41: マージ

- `git merge` **another_branch** [オプション]
- `git merge` **--abort**

パラメーター

パラメータ	
`-m`	マージコミットにめるメッセージ
`-v`	をする
`--abort`	すべてのファイルをのにそうとする
`--ff-only`	マージコミットがなときにちにする
`--no-ff`	マージコミットのをしていなくてもする
`--no-commit`	マージがし、のとができなかった
`--stat`	マージにdiffstatをする
`-n` / `--no-stat`	diffstatをしない
`--squash`	マージされたをつのブランチにして1のコミットをにする

Examples

あるブランチを のブランチにマージする

```
git merge incomingBranch
```

これにより、 のブランチに`incomingBranch`ブランチがマージされます。たとえば、 `master`にいる は、`incomingBranch`が`master`にマージされます。

マージによっては が することがあります。このような は、`Automatic merge failed; fix conflicts and then commit the result.`というメッセージが されます`Automatic merge failed; fix conflicts and then commit the result.` するファイルを で するか、マージの を に すには、 のコマンドを する があります。

```
git merge --abort
```

 マージ

2つのブランチ のコミットが しない 、Gitはそれらを にマージすることができます

```
~/Stack Overflow(branch:master) » git merge another_branch
Auto-merging file_a
```

```
Merge made by the 'recursive' strategy.
 file_a | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
```

マージを□□する

マージを□□した□、マージを□□し、すべてをマージ□の□□に□したい□□があります。 `--abort` □

```
git merge --abort
```

マージの□□からのみ□□を□□する

マージの□に`--ours`または`--theirs`を`git checkout`に`--theirs`て、ファイルのすべての□□をマージの□□または□□から□ることができます。

```
$ git checkout --ours   -- file1.txt # Use our version of file1, delete all their changes
$ git checkout --theirs -- file2.txt # Use their version of file2, delete all our changes
```

コミットでマージする

デフォルトの□□は、マージが□□りとして□□され、マージコミットを□□せずにブランチポインタのみを□□する□□です。□□するには`--no-ff`を□□します。

```
git merge <branch_name> --no-ff -m "<commit message>"
```

□□がマージされていないすべてのブランチを□□する

□□によっては、すでに□□がマスターにマージされているブランチを□に□べていることもあります。 `master`と□□してユニークなコミットを□たない`master`ではないすべてのブランチが□□されます。これは、PRがマスターにマージされた□に□□されなかったブランチを□つけるのに□□に□□です。

```
 for branch in $(git branch -r) ; do
   [ "${branch}" != "origin/master" ] && [ $(git diff master...${branch} | wc -l) -eq 0 ] &&
echo -e `git show --pretty=format:"%ci %cr" $branch | head -n 1`\\t$branch
 done | sort -r
```

オンラインでマージを□む□ https://riptutorial.com/ja/git/topic/291/マージ

42: マージの の

Examples

マニュアル

git mergeを に、 git merge "マージ "エラーを することがあります。どのファイルに があるかを し、 を する があります。

の でgit statusさgit statusば、まだ が なものが に つメッセージで されるのを るのに ちます

```
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")

Unmerged paths:
  (use "git add <file>..." to mark resolution)

    both modified:      index.html

no changes added to commit (use "git add" and/or "git commit -a")
```

Gitはファイルにマーカーを して、 が した を えます

```
<<<<<<<<< HEAD: index.html #indicates the state of your current branch
<div id="footer">contact : email@somedomain.com</div>
========= #indicates break between conflicts
<div id="footer">
please contact us at email@somedomain.com
</div>
>>>>>>>>> iss2: index.html #indicates the state of the other branch (iss2)
```

を するには、<<<<<<と>>>>>>>マーカーの の を に し、ステータス を する があります <<<<<<<、>>>>>>>、そして========= を に。 にgit add index.htmlをgit add index.htmlして したことを し、 git commit をgit commitしてマージを します。

オンラインでマージの の を む https://riptutorial.com/ja/git/topic/3233/マージの の

□ 43: リベース

□□

- `git rebase [-i | --interactive] [options] [--exec <cmd>] [--onto <newbase>] [<upstream>] [<branch>]`
  - `git rebase [-i | --interactive] [options] [--exec <cmd>] [--onto <newbase>] --root [<branch>]`
  - `git rebase --continue | --skip | --abort | --edit-todo`

パラメーター

パラメータ	
-する	マージのをした、リベースプロセスをします。
-アボート	リベースをし、HEADをのブランチにリセットします。リベースがされたときにブランチがされた、HEADはブランチにリセットされます。それの、HEADは、リベースがされたときのにリセットされます。
--keep-empty	その、からもわらないコミットをつ。
-スキップ	のパッチをスキップしてリベースプロセスをします。
-m、--merge	マージをしてリベースする。デフォルトマージをすると、リベースはののをすることができます。リベースマージは、のブランチのにあるブランチからのコミットをすることによってすることにしてください。このため、マージのがした、たちのは、からまってくまでリベースされたシリーズであり、それらのブランチはブランチです。すれば、がれわっている。
--stat	のrebaseからアップストリームにされたもののdiffstatをします。 diffstatは、オプションrebase.statによってもされます。
-x、--exec `command`	のリベースをし、コミットと`command`でする

□□

リベースはリポジトリの□□を□□□に□き□えます。

リモートリポジトリに□□するリベースコミットは、□の□□□がその□□のためのベースノードとして□□しているリポジトリノードを□き□えることができます。あなたが□をしているのか□□に□かっていない□り、□□をプッシュする□にリベースするのがベストプラクティスです。

Examples

ローカルブランチリベース

リベースは、□□のコミットを□のコミットの□に□□□します。

ブランチをrebaseするには、ブランチをチェックアウトし、□のブランチの□にrebaseします。

```
git checkout topic
git rebase master  # rebase current branch onto master branch
```

これにより、

```
      A---B---C topic
     /
D---E---F---G master
```

に□える□

```
             A'--B'--C' topic
            /
D---E---F---G master
```

これらの□□を1つのコマンドにまとめると、ブランチをチェックアウトして□ちにリベースすることができます。

```
git rebase master topic   # rebase topic branch onto master branch
```

□□□リベース□、□□されたコミットは□なるハッシュを□ちます。すでにリモートホストにプッシュしたコミットを□□□しないでください。□□はできないかもしれgit pushためにあなたの□□のオプション□して、リモートホストにローカルのリベース□のブランチをgit push --force。

Rebase□□たちと□らの、ローカルとリモート

リベースは「□たち」と「□ら」の□□を□えます□

```
git checkout topic
git rebase   master    # rebase topic branch on top of master branch
```

HEADの□しているものが"□たち"

リベースが□□に□うことは、HEADをmasterリセットすることです。チェリーピッキングが□いブランチtopicから□しいブランチtopicにコミットする□に□□のtopicブランチのコミットはすべて□き□えられ、□のハッシュによって□□されます□。

マージツールで□□される□□□ ローカルリファレンスまたはリモートリファレンスと□□しないでください□に□しては、

```
=> local is master ("ours"),
=> remote is topic ("theirs")
```

つまり、マージ/□□ツールは、local □ master □リベースしているブランチ□と□□ブランチをremoteとして□□します□ topic □リベースされているブランチ□

```
+-----------------------------------------+
| LOCAL:master |    BASE    | REMOTE:topic |
+-----------------------------------------+
|              MERGED                     |
+-----------------------------------------+
```

イラストレーション

マージ□□

```
c--c--x--x--x(*) <- current branch topic ('*'=HEAD)
    \
     \
      \--y--y--y <- other branch to merge
```

□たちは□□の□□のtopic□□しないので、□たちはまだ□たちが□り□んでいるものです□そして□たちは□の□□から□□します□

```
c--c--x--x--x---------o(*)  MERGE, still on branch topic
    \       ^        /
     \     ours     /
      \            /
       --y--y--y--/
              ^
            theirs
```

リベースの□□□

しかし、リベースでは、リベースが□□に□うことは、□□のブランチをチェックアウトして、その□にある□□のコミットを□□するためです。

```
c--c--x--x--x(*) <- current branch topic ('*'=HEAD)
    \
     \
      \--y--y--y <- upstream branch
```

git rebase upstreamの**git rebase upstream**は、まずHEADを□□のブランチに□□します。したがって、□□の□□の□□ブランチと□□して、 '□たち'と '□□のもの'の□り□えが□われます。

```
c--c--x--x--x <- former "current" branch, new "theirs"
    \
     \
      \--y--y--y(*) <- set HEAD to this commit, to replay x's on it
               ^       this will be the new "ours"
               |
           upstream
```

rebaseは□しい '□たちの' topicブランチで '□らの'コミットを□□します□

```
c--c..x..x..x <- old "theirs" commits, now "ghosts", available through "reflogs"
    \
     \
      \--y--y--y--x'--x'--x'(*) <- topic  once all x's are replayed,
               ^                      point branch topic to this commit
               |
         upstream branch
```

インタラクティブリベース

この□は、□□モードでgit rebaseどのように□□できるかを□□することを□□としています。 git rebaseが□であるか、それが□をするのかを□□□に□□していることが□□されます。

□□□のリベースは、□のコマンドを□□して□□されます。

```
git rebase -i
```

`-i`オプションは□□モードを□します。□□□のリベースを□□すると、ユーザーはコミットメッセージを□□したり、□べ□え、□□、および/またはスカッシュ□1つに□□□コミットを□□することができます。

□□の3つのコミットを□□□したいとします。これを□□するには、□□を□□します。

```
git rebase -i HEAD~3
```

□□の□□を□□した□、テキストエディタでファイルを□き、コミットのリベース□□を□□することができます。この□では、コミットの□□を□□し、ファイルを□□してエディタを□じます。これにより、□□した□□でリベースが□□されます。 `git log`をチェックすると、□□した□しい□□でコミットが□□されます。

□コミット メッセージ

さて、コミットメッセージの1つがあいまいであり、それをより□□□にしたいと□めました。□じコマンドを□って□□の3つのコミットを□べてみましょう。

```
git rebase -i HEAD~3
```

コミットの□□を□べ□える□わりに、コミットは`reword`れます。□□は、デフォルトの`pick`を□□して、コミット□にメッセージを□□したい□□に`reword`します。

エディタを□じると、リベースが□□され、リワードする□□のコミットメッセージで□□します。これにより、コミットメッセージをあなたが□むものに□□することができます。メッセージを□□したら、エディタを□じて□□します。

コミットの□□の□□

コミットメッセージの□□に□えて、コミットによって□われた□□を□□させることもできます。これを□うには、1つのコミットに□して`edit`ために`pick`を□□し`edit`。 Gitは、そのコミットに□□すると□□し、ステージング□□でのコミットの□の□□を□□します。これらの□□は、ステージングを□□したり、□しい□□を□えたりすることで、□□できます。

ステージング□□にコミットに□□なすべての□□が□まれるとすぐに、□□をコミットします。□いコミットメッセージが□□され、□しいコミットを□□させることができます。

□□のコミットを□□に□□する

コミットしたとしますが、□でこのコミットを2つ□□のコミットに□□することを□□しました。□□と□じコマンドを□□して、□わりに`edit`を`pick`に□き□え、`Enter`を□します。

さて、`git`はあなたが□□□にマークしたコミットで□□し、すべてのコンテンツをステージングエリアに□□します。その□□から`git reset HEAD^`を□□して、コミットを□□ディレクトリに□くことができます。□に、ファイルを□の□□で□□してコミットすることができます。□□□に、1□のコミットを*n*□のコミットに□□します。

□□のコミットを1つにまとめる

いくつかの□□をして、1□のコミットではないと□われる□□のコミットがあるとします。そのためには`git rebase -i HEAD~3`を□□し、`3`を□□な□のコミットに□き□えてください。

□□は`pick`を`squash`に□き□えてください。 `rebase`□に、あなたが□しつぶされるように□□したコミットは、□のコミットの□で□しつぶされます。□わりにそれらを□□のコミットに□えます。

インタラクティブな**Rebase**を□□する

□□□のリベースを□□しました。コミットを□□するエディタで、□かが□□っていると□□した□□□たとえば、コミットがない、または□□ったリベース□を□□したなど□、リベースを□□したいとします。

これを□うには、すべてのコミットとアクション□つまり、 `#□□`で□まらない□□をすべて□□すると、リベースが□□されます□

エディタのヘルプテキストは□□にこのヒントを□□します□

```
# Rebase 36d15de..612f2f7 onto 36d15de (3 command(s))
#
# Commands:
# p, pick = use commit
# r, reword = use commit, but edit the commit message
# e, edit = use commit, but stop for amending
# s, squash = use commit, but meld into previous commit
# f, fixup = like "squash", but discard this commit's log message
# x, exec = run command (the rest of the line) using shell
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
#          ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
# Note that empty commits are commented out
```

リベ—ス□のプッシュ

ときには、リベ—スを□ってリライトの□□を□き□す□□があるかもしれませんが、 `git push`は□□を□き□えたのでそうすることについて□□を□います。

これは`git push --force`で□□できますが、 `git push --force-with-lease`□□すると、ロ—カルのリモ—ト□□ブランチがリモ—トのブランチと□なる□□□□えば、□か□、プッシュを□□させたいことを□しますそれ□□は□□のフェッチ□にリモ—トにプッシュされます。これにより、□の□かの□□のプッシュを□って□□きすることを□ぎます。

□□ □ `git push --force` –とさえ`--force-with-lease`そのことについては-それはブランチの□□を□き□えているため、□□なコマンドであることができます。□の□が□□□にプッシュする□にブランチを□っ□った□□、ロ—カル□□とリモ—ト□□が□なるため、 `git pull`または`git fetch`にエラ—が□□します。これにより、□に□□しないエラ—が□□する□□□があります。 `reflog`を□□に□れば、□のユ—ザ—の□□を□□することができますが、□□が□□される□□□があります。□のコントリビュ—タとブランチに□□□にプッシュする□□がある□□は、エラ—を□□する□□がないようにコ—ディネ—トしてください。

□□のコミットまで□げる

Git 1.7.12□□、ルートコミットにリベ—スできます。ルートコミットはリポジトリで□めて□□されたコミットであり、□□は□□できません。□のコマンドを□□します。

```
git rebase -i --root
```

コ—ドレビュ—の□にリベ—ス

□□

この□□は、□らばったすべてのコミットをより□□のあるコミットに□□□して、□□なコードレビュ—を□うことです。□□に□くのファイルに□□のレイヤ—が□すぎる□□は、コードレビュ—を□うのが□しくなります。□□□□に□□されたコミットをトピックコミットに□□□することができれば、コードレビュープロセスはより□□になります□そして、おそらくコードレビュープロセスのバグは□なくなります□。

この□□に□□□された□だけでは、`git`を□□してコードレビュ—を□□する□□の□□ではありません。これは□がやる□□であり、コードレビュ—と`git`の□□をより□□に/より□くする□□を□の□に□□するものです。

これは□□□にリベ—スの□を□□□にも□しています。

この□では、□□□のリベ—スについて□っていることを□□としています

□□□

- あなたはマスタ—のフィ—チャ—ブランチに□り□んでいます
- あなたの□□は3つの□□な□を□っています□フロントエンド、バックエンド、DB
- フィ—チャ—ブランチで□□している□にたくさんのコミットを□っています。□コミットは□□に□□のレイヤ—に□れる
- あなたは□□□に□ブランチに3つのコミットしか□みません
    - 1つはすべてのフロントエンドの□□を□む
    - 1つはすべてのバックエンド□□を□む
    - 1つはすべてのDB□□を□む

□□□

- □たちは□□□コミットを「□□□」コミットに□□しようとしています。
- まず、すべてのコミットを□□の□さなコミットに□□します。□コミットは、□□に1つのトピックのみを□んでいます□この□では、トピックはフロントエンド、バックエンド、DB□□です□
- □に、□たちの□□□なコミットを□べ□えて、それらを□□の□□□なコミットにスカッシュします

□□

```
$ git log --oneline master..
975430b db adding works: db.sql logic.rb
3702650 trying to allow adding todo items: page.html logic.rb
43b075a first draft: page.html and db.sql
$ git rebase -i master
```

これはテキストエディタで□□されます□

```
pick 43b075a first draft: page.html and db.sql
pick 3702650 trying to allow adding todo items: page.html logic.rb
pick 975430b db adding works: db.sql logic.rb
```

これに□□してください□

```
e 43b075a first draft: page.html and db.sql
e 3702650 trying to allow adding todo items: page.html logic.rb
e 975430b db adding works: db.sql logic.rb
```

□にgitは□□に1つのコミットを□□します。□コミット□、プロンプトが□□され、□のことができます。

```
Stopped at 43b075a92a952faf999e76c4e4d7fa0f44576579... first draft: page.html and db.sql
You can amend the commit now, with

        git commit --amend

Once you are satisfied with your changes, run

        git rebase --continue

$ git status
rebase in progress; onto 4975ae9
You are currently editing a commit while rebasing branch 'feature' on '4975ae9'.
  (use "git commit --amend" to amend the current commit)
  (use "git rebase --continue" once you are satisfied with your changes)
```

```
nothing to commit, working directory clean
$ git reset HEAD^ #This 'uncommits' all the changes in this commit.
$ git status -s
 M db.sql
 M page.html
$ git add db.sql  #now we will create the smaller topical commits
$ git commit -m "first draft: db.sql"
$ git add page.html
$ git commit -m "first draft: page.html"
$ git rebase --continue
```

その□、コミットごとにこれらの□□を□り□します。□□、あなたはこれを□っています□

```
$ git log --oneline
0309336 db adding works: logic.rb
06f81c9 db adding works: db.sql
3264de2 adding todo items: page.html
675a02b adding todo items: logic.rb
272c674 first draft: page.html
08c275d first draft: db.sql
```

□□はもう□□rebaseを□□して、□べ□えとスカッシュを□□します。

```
$ git rebase -i master
```

これはテキストエディタで□□されます□

```
pick 08c275d first draft: db.sql
pick 272c674 first draft: page.html
pick 675a02b adding todo items: logic.rb
pick 3264de2 adding todo items: page.html
pick 06f81c9 db adding works: db.sql
pick 0309336 db adding works: logic.rb
```

これに□□してください□

```
pick 08c275d first draft: db.sql
s 06f81c9 db adding works: db.sql
pick 675a02b adding todo items: logic.rb
s 0309336 db adding works: logic.rb
pick 272c674 first draft: page.html
s  3264de2 adding todo items: page.html
```

□□□git rebase に、□□□にコミットされた□さなコミットを□□/ □□するように□□してください 。さもなければ、□□の□□なマ—ジ□□が□□する□□□があります。

そのィンタラクティブ·リベ—スがすべて□われ、□□されると、あなたはこれを□ます□

```
$ git log --oneline master..
74bdd5f adding todos: GUI layer
e8d8f7e adding todos: business logic layer
121c578 adding todos: DB layer
```

□□

あなたは□□□コミットを□□□コミットにリベ—スしました。□□の□□では、これを□□□う□□はないかもしれませんが、あなたがこれをやりたい、またはする□□があるときに、□できるのです。さらに、うまくいけば、git rebaseについてもっと□んだことでしょう。

マージの□わりに□□□に**rebase**を□□するように**git-pull**をセットアップする

チームがリベースベースのワークフローに□っている□□、 `git pull`□に、□しく□□された□ブランチがマージ□□ではなくリベース□□を□□するようにgitをセットアップすることが□□かもしれません。

すべての□しいブランチを□□□に.gitconfigするように□□するには、 .gitconfigまたは.git/config□の□を□□します。

```
[branch]
autosetuprebase = always
```

コマンドライン□ *git config [--global] branch.autosetuprebase always*

あるいは、-- --rebaseオプションが□されたかのように□に□□するようにgit pullコマンドを□□することもできます□

```
[pull]
rebase = true
```

コマンドライン□ *git config [--global] pull.rebase true*

リベース□のすべてのコミットをテストする

プルリクエストを□う□に、コンパイルが□□し、ブランチ□の□コミットに□してテストが□□していることを□□すると□□です。 `-x` パラメータを□って□□□に□うことができます。

□えば□

```
git rebase -i -x make
```

□□□なrebaseを□□し、 makeを□□makeたびに□□します。 makeが□□した□□、gitは□□して、□□を□□し、□のものを□ぶ□にコミットを□□する□□を□えます。

オートスタートの□□

Autostashは、ローカル□□にリベースを□□する□□に□□に□□な□□オプションです。□くの□□、□□のブランチからコミットする□□があるかもしれませんが、まだコミットする□□ができていません。

しかし、Gitでは、□□ディレクトリがクリーンでなければ、リベースを□□できません。レスキューへのオートスタート□

```
git config --global rebase.autostash    # one time configuration
git rebase @{u}                         # example rebase on upstream branch
```

□□□が□□すると、□□□□が□□されます。リベースが□□に□□したかどうか、または□□□□したかどうかは□□ありません。どちらの□□でも、オートスタッシュが□□されます。リベースが□□し、ベースコミットが□□された□□、□□ストッシュと□しいコミットの□に□□が□□する□□□があります。この□□、コミットする□に□□を□□する□□があります。これは、□□で□してから□□する□□と□わりありませんので、□□□に□うことには□□はありません。

オンラインでリベースを□む□ https://riptutorial.com/ja/git/topic/355/リベース

□ 44: リポジトリのクローニング

□□

- `git clone [<options>] [ - ] <repo> [<dir>]`
- `git clone [--template = <template_directory>] [-l] [-s] [--no-hardlinks] [-q] [-n] [--bare] [ - ミラー] [-o <□□>リポジトリ>] [--dissociate] [-separate-git-dir <gitディレクトリ>] [--depth <□さ> ] [-[no-] single-branch] [--recursive | - [サブディレクトリ] - [いいえ]□いサブモジュール] - [ジョブ] [ - ] <リポジトリ> [<ディレクトリ>]`

Examples

□いクローン

□□なリポジトリ□□□の□□を□つプロジェクトなど□を□□するには、□□するデータの□が□いために□□がかかるか□□する□□□があります。□□な□□を□□する□□がない□□は、□いクローンを□□できます。

```
git clone [repo_url] --depth 1
```

□□のコマンドは、リモートリポジトリからの□□のコミットだけをフェッチします。

□いリポジトリでのマージを□□できない□□があることに□□してください。マージを□□するためにバックトラックする□□があるのは、□なくとも□くのコミットを□ることをお□めします。たとえば、□□に50□のコミットを□□するには、□のようにします。

```
git clone [repo_url] --depth 50
```

□で□□に□じてリポジトリの□りの□□を□□することができます□

1.8.3

```
git fetch --unshallow     # equivalent of git fetch --depth=2147483647
                          # fetches the rest of the repository
```

1.8.3

```
git fetch --depth=1000    # fetch the last 1000 commits
```

レギュラークローン

□□な□□とすべてのブランチを□むリポジトリ□□をダウンロードするには、□のように□□します。

```
git clone <url>
```

□□の□では、リポジトリ□と□じ□□のディレクトリに□□します。

リポジトリをダウンロードして□□のディレクトリに□□するには、□のように□□します。

```
git clone <url> [directory]
```

□□については、 リポジトリのクローニングを□□してください。

□□のブランチをクローンする

リポジトリの□□のブランチを□□するには、リポジトリのURLの□に`--branch <branch name>`します。

```
git clone --branch <branch name> <url> [directory]
```

--branchに□□□オプションを□□するには、 -bと□□し-b 。このコマンドはリポジトリ□□をダウンロードし、 <branch name> をチェックアウトします。

ディスクスペースを□□するために、□□を□□して□□のブランチにのみつながる□□をクローンすることができます□

```
git clone --branch <branch_name> --single-branch <url> [directory]
```

--single-branchコマンドに□□しないと、すべてのブランチの□□が[directory]□□されます。これは□きなリポジトリで□□になる□□□があります。

□で--single-branchフラグを□り□し、□りのリポジトリ□□コマンドをフェッチするには□

```
git config remote.origin.fetch "+refs/heads/*:refs/remotes/origin/*"
git fetch origin
```

□□□にクローンを□□する

1.6.5

```
git clone <url> --recursive
```

リポジトリをクローンし、すべてのサブモジュールをクローンします。サブモジュール□□に□□のサブモジュールが□まれている□□、Gitはそれらもクローンします。

プロキシを□□したクローン□□

gitでプロキシでファイルをダウンロードする□□がある□□は、プロキシサーバをシステム□□に□□するだけでは□□ではありません。あなたはまた、□□を□すことができます□

```
git config --global http.proxy http://<proxy-server>:<port>/
```

オンラインでリポジトリのクローニングを□む□ https://riptutorial.com/ja/git/topic/1405/リポジトリのクローニング

□ 45: リモートでの□□

□□

- git remote [-v | --verbose]
  - git remote add [-t <branch>] [-m <master>] [-f] [--[no-]tags] [--mirror=<fetch|push>] <name> <url>
  - git remote rename <old> <new>
  - git remote remove <name>
  - git remote set-head <name> (-a | --auto | -d | --delete | <branch>)
  - git remote set-branches [--add] <name> <branch>…
  - git remote get-url [--push] [--all] <name>
  - git remote set-url [--push] <name> <newurl> [<oldurl>]
  - git remote set-url --add [--push] <name> <newurl>
  - git remote set-url --delete [--push] <name> <url>
  - git remote [-v | --verbose] show [-n] <name>…
  - git remote prune [-n | --dry-run] <name>…
  - git remote [-v | --verbose] update [-p | --prune] [(<group> | <remote>)…]

Examples

□しいリモートリポジトリを□□する

```
git remote add upstream git-repository-url
```

git-repository-url□されるリモートのgitリポジトリをupstream□しいリモート□としてgitリポジトリに□□します

アップストリームリポジトリからの□□

アップストリームを□□すると□□します□「□□リポジトリの□□」のように□

```
git fetch remote-name
git merge remote-name/branch-name
```

pullコマンドは、 fetchとmergeます。

```
git pull
```

pullと--rebase flagコマンドを□み□わせてfetchとrebaseの□わりにmerge。

```
git pull --rebase remote-name branch-name
```

ls-remote

git ls-remoteは、□□にクローン/フェッチすることなく 、リモートのリポジトリに□い□わせることを□□にする□□のコマンドです。

リモートリポジトリのrefs / headsとrefs / tagsを□□□□します。

□□□された□□□きタグ□すなわち、そのタグが□しているコミット□をリストするために、□々 refs/tags/v0.1.6 と refs/tags/v0.1.6^{} □ ^{}

git 2.8□2016□3□□□、タグの□□□□を□けることができ、□□のように□□されたタグを□□リストすることができます□

```
git ls-remote --ref
```

また、"url.<base>.insteadOf "□□がある□□、リモートリポジトリで□□される□□のURLを□□するのに□□ちます。
git remote --get-url <aremotename>がhttps://server.com/user/repoを□し、 git config url.ssh://git@server.com:.insteadOf https://server.com/ □

```
git ls-remote --get-url <aremotename>
ssh://git@server.com:user/repo
```

リモートブランチの□□

Gitのリモートブランチを□□するには□

```
git push [remote-name] --delete [branch-name]
```

または

```
git push [remote-name] :[branch-name]
```

□□されたリモートブランチのローカルコピーの□□

リモートブランチが□□されている□□は、ローカルリポジトリに□□をプルーニングするように□□する□□があります。

□□のリモートから□□されたブランチを□□するには□

```
git fetch [remote-name] --prune
```

すべてのリモートから□□されたブランチを□□するには□

```
git fetch --all --prune
```

□□のリモートに□する□□を□□する

□□のremote□ originについての□□を□□する

```
git remote show origin
```

リモートのURLだけを□□する□

```
git config --get remote.origin.url
```

2.7+では、これも□□ですが、これは□□のconfigコマンドを□□するよりはるかに□れています。

```
git remote get-url origin
```

□□のリモコンを□□□□する

このリポジトリに□□□けられている□□のリモートをすべて□□□□します。

```
git remote
```

このリポジトリに□□□けられている□□のすべてのリモコンを、 fetch URLとpush URLを□めて□しく□□□□します。

```
git remote --verbose
```

または□に

```
git remote -v
```

□□

リモートブランチにプッシュするための□□

```
git push <remote_name> <branch_name>
```

□

```
git push origin master
```

□しい□□で□□を□□する

□しいブランチを□□し、それを□□してブランチに□り□えることができます

```
git checkout -b AP-57
```

`git checkout`を□□して□しいブランチを□□した□は、その□□の□□を□□して□□する□□があります

```
git push --set-upstream origin AP-57
```

その□、そのブランチにいる□に`git push`を□うことができます。

リモートリポジトリの□□

リモートが□し□すリポジトリのURLを□□するには、□のように`set-url`オプションを□□します□

```
git remote set-url <remote_name> <remote_repository_url>
```

□□

```
git remote set-url heroku https://git.heroku.com/fictional-remote-repository.git
```

Gitリモート**URL**を□□する

□□のリモートを□□する

```
git remote -v
# origin https://github.com/username/repo.git (fetch)
# origin https://github.com/usernam/repo.git (push)
```

リポジトリURLの□□

```
git remote set-url origin https://github.com/username/repo2.git
# Change the 'origin' remote's URL
```

□しいリモートURLを□□する

```
git remote -v
# origin  https://github.com/username/repo2.git (fetch)
# origin  https://github.com/username/repo2.git (push)
```

リモートの□□を□□する

リモートの□□を□□するには、コマンド`git remote rename`□□します。

`git remote rename`コマンドは2つの□□をとります□

- □□のリモート□□□□ **origin**□
- リモートの□しい□□です□□□ **destination**□。

□□のリモート□を□□する

```
git remote
# origin
```

□□のリモートを`URL`で□□する

```
git remote -v
# origin https://github.com/username/repo.git (fetch)
# origin https://github.com/usernam/repo.git (push)
```

リモートの□□を□□する

```
 git remote rename origin destination
 # Change remote name from 'origin' to 'destination'
```

□しい□□を□□する

```
git remote -v
# destination https://github.com/username/repo.git (fetch)
# destination https://github.com/usernam/repo.git (push)
```

===□□なエラー**===**

1. □□セクション'`remote`。[old name]'の□□を'`remote`。[new name]'に□□できませんでした

   このエラーは、□いリモート□□ □□ □を□したリモートが□□しないことを□□します。

2. リモート[□しい□□]は□に□□します。

   エラーメッセージは□□です。

□□のリモートの**URL**を□□する

コマンドで□□のリモートの`URL`を□□することができます

```
git remote set-url remote-name url
```

□□のリモートの**URL**を□□する

このコマンドを□□すると、□□のリモートの`URL`を□□できます

```
git remote get-url <name>
```

デフォルトでは、これは

```
git remote get-url origin
```

オンラインでリモートでの□□を□む□ https://riptutorial.com/ja/git/topic/243/リモートでの□□

□ 46: ワークツリー

□□

- `git worktree add [-f] [--detach] [--checkout] [-b <new-branch>] <path> [<branch>]`
- `git worktree prune [-n] [-v] [--expire <expire>]`
- `git worktree list [--porcelain]`

パラメータ

パラメータ	
-f --force	デフォルトでは、`<branch>`がのツリーによってすでにチェックアウトされているときに、しいツリーをすることをします。このオプションは、そのセーフガードをにします。
-b `<new-branch>` -B `<new-branch>`	アドオンでは、というのしいブランチを`<new-branch>`からまる`<branch>`、およびチェックアウト`<new-branch>`しいツリーに。`<branch>`がされた、デフォルトは`HEAD`ます。では、`-b`、すでにするはしいブランチのをします。`-B`このセーフガードをにし、`<new-branch>`を`<branch>`にリセットします。
--detach	すると、しいツリーに`HEAD`をデタッチします。
- [no-]チェックアウト	デフォルトでは、`<branch>` add `<branch>`は`<branch>`チェックアウトしますが、`--no-checkout`をしてチェックアウトをして、スパースチェックアウトのなどのカスタマイズをうことができます。
-n --dry-run	プルーンでは、もしないでください。それがするものだけをしてください。
-	リストをすると、スクリプトのがなでされます。このフォーマットは、Gitバージョンで、ユーザーのになくしています。
-v --verbose	で、すべてのをしてください。
--expire `<time>`	プルーンでは、`<time>`よりいのツリーをれにします。

□□

□□については、□□ドキュメントを□□してください□ https : //git-scm.com/docs/git-worktree。

Examples

ワークツリーの□□

あなたは□しい□□の□□の□っ□□です。あなたの□□はすぐに□かを□□するよう□□しています。□□は、`git stash`を□□して□□を□□□に□□することができます。しかし、□□□では、□□ツリーは□□している□□しいファイル、□□されたファイル、□□されたファイル、その□のビットとピースが□□している□ので、□□を□げたくはありません。

ワークツリーを□□することによって、□□□にリンクされた□□ツリーを□□して□□□□を□い、□□したら□□して□□のコーディングセッションを□□します。

```
$ git worktree add -b emergency-fix ../temp master
$ pushd ../temp
# ... work work work ...
$ git commit -a -m 'emergency fix for boss'
$ popd
$ rm -rf ../temp
$ git worktree prune
```

□□この□では、□□プログラムはまだ□□□□ブランチにあります。この□□で、 `git merge`または`git format-patch`を□□した□、□□□□ブランチを□□することをお□めします。

ワークツリーの□□

□□□バージョン2.11.0□□□、□□のワークツリーを□□するための□み□み□□はありません。これは□□のバグとして□□されています □ https://git-scm.com/docs/git-worktree#_bugs を□□□ 。

この□□を□□するために、 `.git`□□ファイルで□□□□□□を□□することは□□です。

この□では、`repo`のメインコピーは`/home/user/project-main`あり、セカンダリワークツリーは`/home/user/project-1`あり、 `/home/user/project-2`。

これらのステップの□で`git`コマンドを□□しないと、ガベージコレクタが□□され、セカンダリツリーへの□□が□われる□□□があります。□□することなく、□□から□□まで□の□□を□□します。

1. ワークツリーの`.git`ファイルを□□して、メインツリー□の□しい□□を□すようにします。 `/home/user/project-1/.git`ファイルに□の□□が□まれているはずです□

   ```
   gitdir: /home/user/project-main/.git/worktrees/project-2
   ```

2. メインワークブックの`.git`ディレクトリ□にあるワークツリーの□□を□□します。

   ```
   $ mv /home/user/project-main/.git/worktrees/project-1 /home/user/project-main/.git/worktrees/project-2
   ```

3. `/home/user/project-main/.git/worktrees/project-2/gitdir`□の□□を□しい□□を□すように□□します。この□では、ファイルの□□は□のようになります。

   ```
   /home/user/project-2/.git
   ```

4. □□に、□□□□を□しい□□に□□します。

   ```
   $ mv /home/user/project-1 /home/user/project-2
   ```

すべてを□しく□った□□は、□□のワークツリーをリストすると、□しい□□を□□する□□があります。

```
$ git worktree list
/home/user/project-main  23f78ad [master]
/home/user/project-2     78ac3f3 [branch-name]
```

`git worktree prune`を□□するのも□□です。

オンラインでワークツリーを□む□ https://riptutorial.com/ja/git/topic/3801/ワークツリー

□ 47: ワークフローのタイプの□□

□□

Gitのようなバージョン□□ソフトウェアを□うと、□□は□し□いかもしれませんが、□□に□□した□□□な□□は、さまざまな□□のワークフローを□□にします。あなた□□の□□チームに□したものを□んでください。

Examples

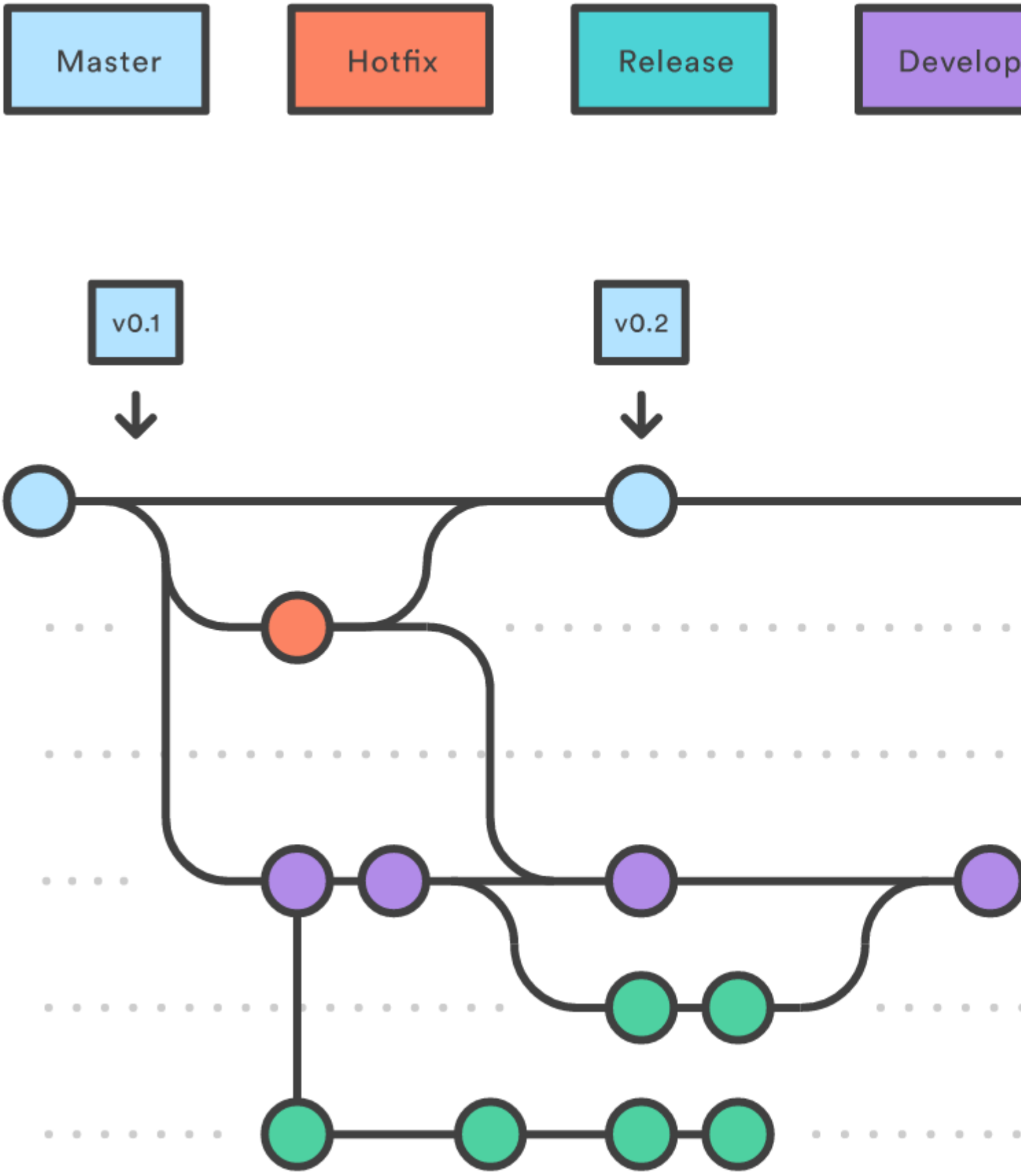
Gitflowのワークフロー

もともとVincent Driessenによって□□されたGitflowは、gitといくつかの□□□□ブランチを□□した□□ワークフローです。これは、 Feature Branch Workflowの□□なケースと□なすことができます。

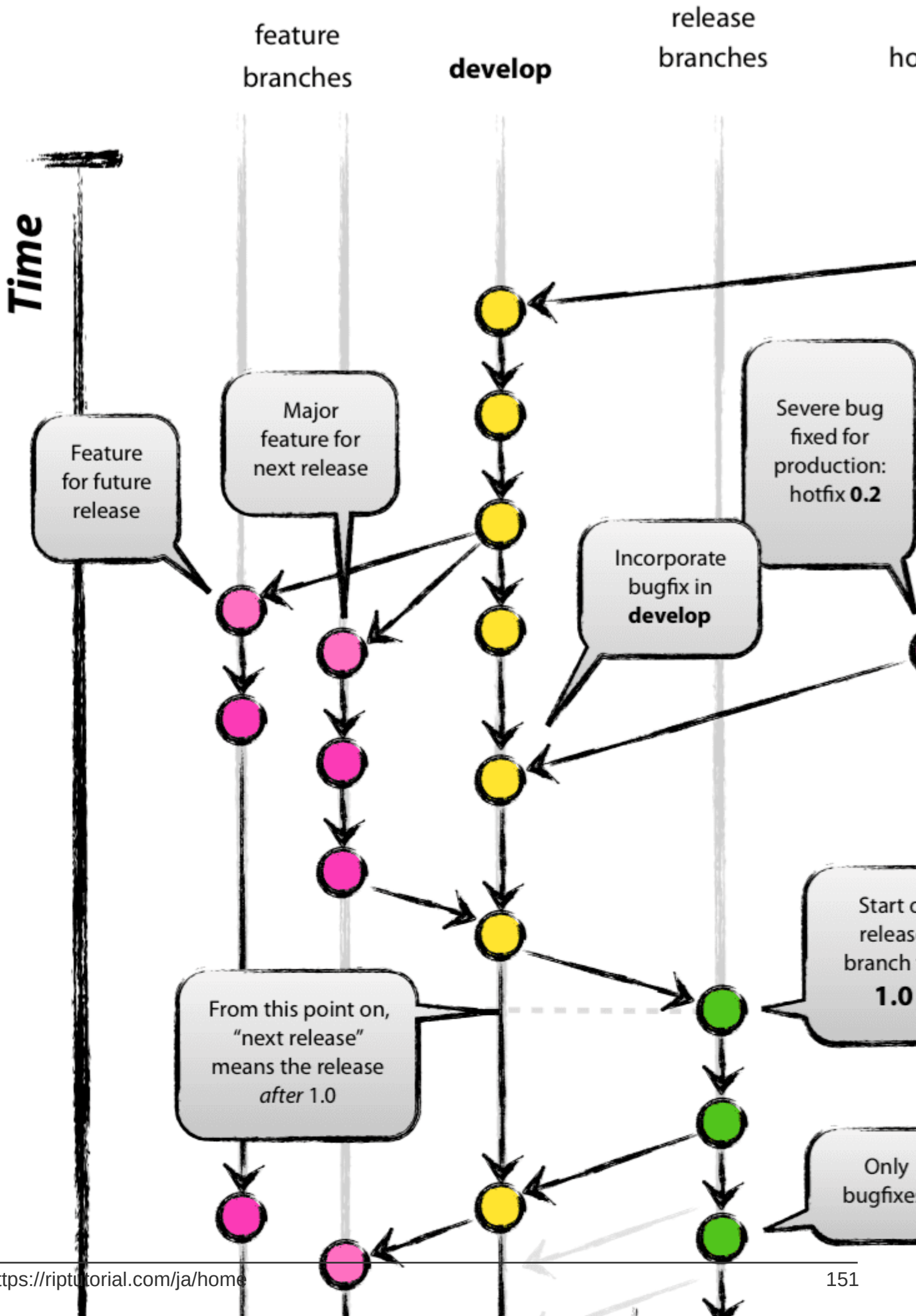
この1つのアイデアは、□□□の□□の□□□に□々のブランチを□□することです。

- masterブランチは□に□□のプロダクションコードです。□□コードはここに□しません。
- developブランチには□□の□□がすべて□まれています。これらの□□□な□□はほとんど□でもあり□ますが、より□きな□□はそれぞれのブランチのために□□されています。ここのコードはリリース/デプロイメント□に□に□□され、 releaseに□□されます。
- hotfixブランチは、□のリリースまで□つことのできない□さなバグの□□□です。 hotfixブランチはmasterから□れ、masterとdevelop□□にマージされます。
- releaseブランチは、 developからmaster□しい□□をリリースするために□□されます。バンプするバージョン□□などの□□の□□は、リリースブランチで□われ、その□、マージされてmaster□され、 developます。□しいバージョンをデプロイするときは、□□の□□と□□なロールバックのために、 master□□のバージョン□□□たとえば、 セマンティックバージョニング □をタグ□けする□□があります。
- featureブランチは、より□きな□□のために□□されています。これらは、□に□□された□□で□□されて□□□されているdevelop□えたとき。□□のfeatureブランチは、□□を□□し、 □□した□□を□いに□□して□□できるようにします。

このモデルの□□□□□□

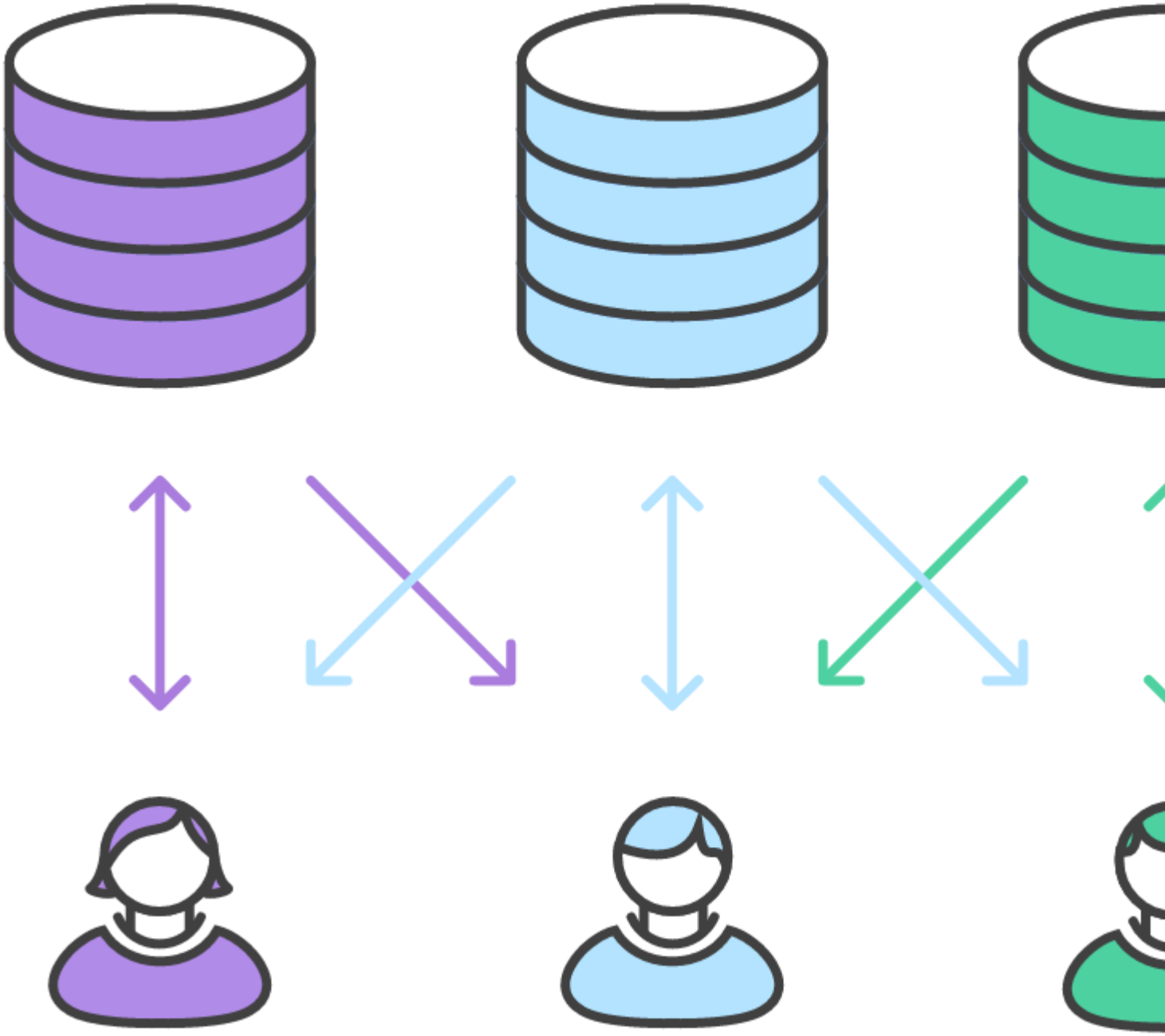


このモデルの の


feature branches
develop
release branches
Time
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
From this point on, "next release" means the release after 1.0

できる1つの□□ リポジトリを□ つ□ わりに、□□□□はメインリポジトリから□□ した□□のリポジトリを□っています。これの□□は、□□□が□□ リポジトリではなく、□□のreposに□□できき、□□□は、□□に□じて、フォークされたリポジトリからの□□ をオリジナルに□□できることです。

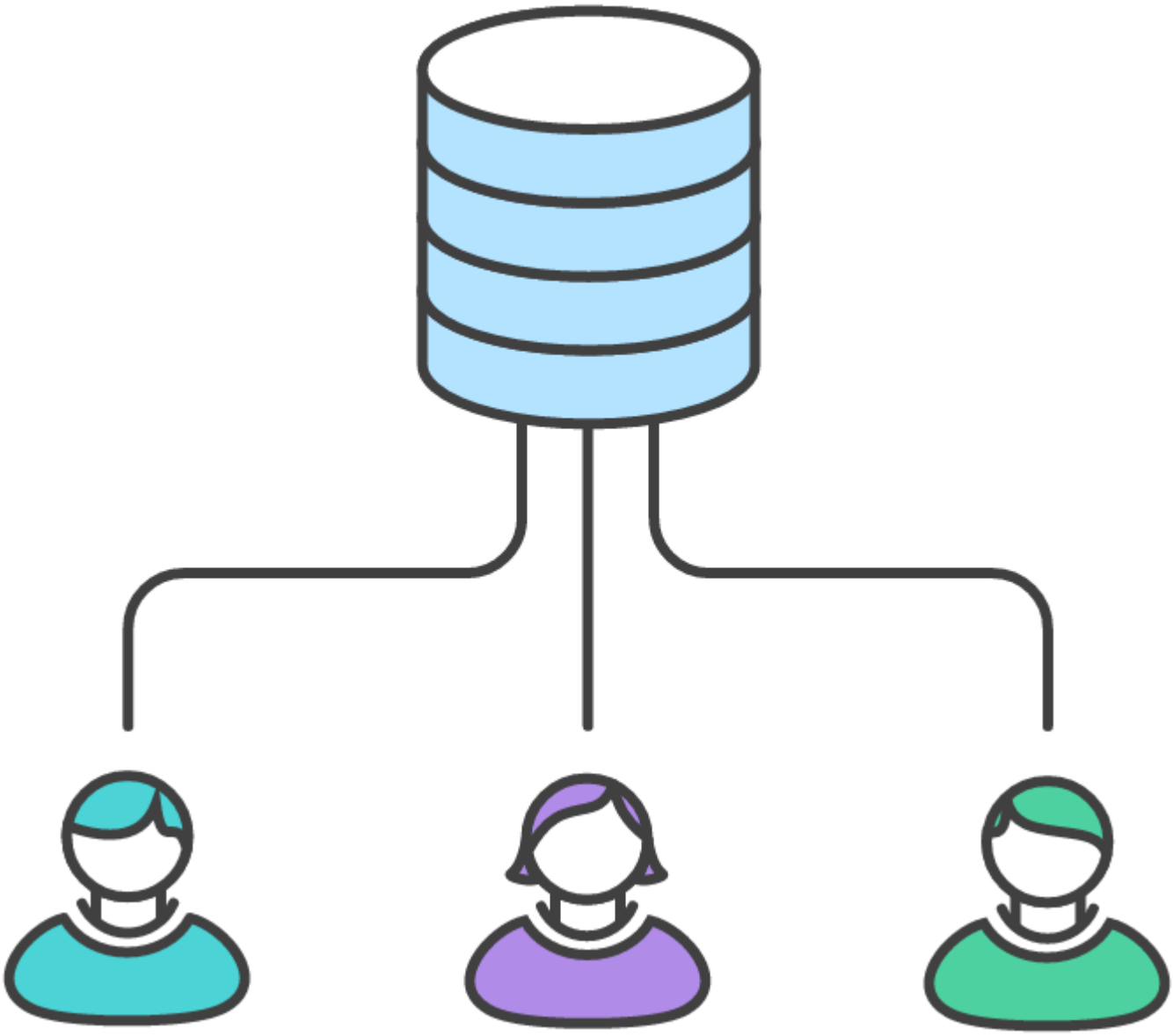
このワークフローの□□□□□は□のとおりです。



□□ ワークフロー

この□□□なワークフローモデルでは、 masterブランチにはすべてのアクティブな□□が□まれています。□□□は、□□ を□□ する□に□□の□□ を□ り□れることを□に□□ する□□ があります。このブランチは□□に□□ するためです。□もがこのリポジトリにアクセスし、□□をマスターブランチにコミットできます。

このモデルのビジュアル□□□



これは、SubversionやCVSのような□いシステムが□□された□□□なバージョン□□のパラダイムです。このように□□するソフトウェアは、□□バージョン□□システム□CVCS□と□ばれます。 Gitはこのように□□することができますが、すべてのプルにマージを□□させる□□があるなど、□きな□□があります。チームがこのように□□することは□□に□□ですが、□□のマージ□□の□□によって□くの□□な□□が□かることになります。

これが、Linus TorvaldsがGitを*CVCS*としてではなく、Mercurialに□た*DVCS*、または*Distributed Version Control System*として□った□□です。この□しい□□の□□は、このページの□の□で□されている□□□です。

□□ブランチワークフロー

□□ブランチワークフローの□□にある□□なアイデアは、すべての□□□□がmasterブランチではなく□□ブランチで□われることです。このカプセル□により、□□の□□□が□□なコードベースを□すことなく、□□の□□を□□に□□できます。また、 masterブランチには□れたコードが□まれることはなく、これは□□□な□□□□にとって□きな□□です。

□□の□□をカプセル□することで、プルリクエストを□□することも□□になります。これは、ブランチの□りでディスカッションを□□する□□です。□らは□の□□□に、□□プロジェクトに□□される□に□□を□□する□□を□えます。また、□□の□□で□ち□□した□□は、□□からの□□を□めるプルリクエストを□くことができます。□□は、プルリクエストによって、あなたのチームがお□いの□□についてコメントすることが□□に□□になります。

Atlassianチュートリアルに□づいています。

GitHubの□れ

□くのオープンソースプロジェクトで□□がありますが、それだけではありません。

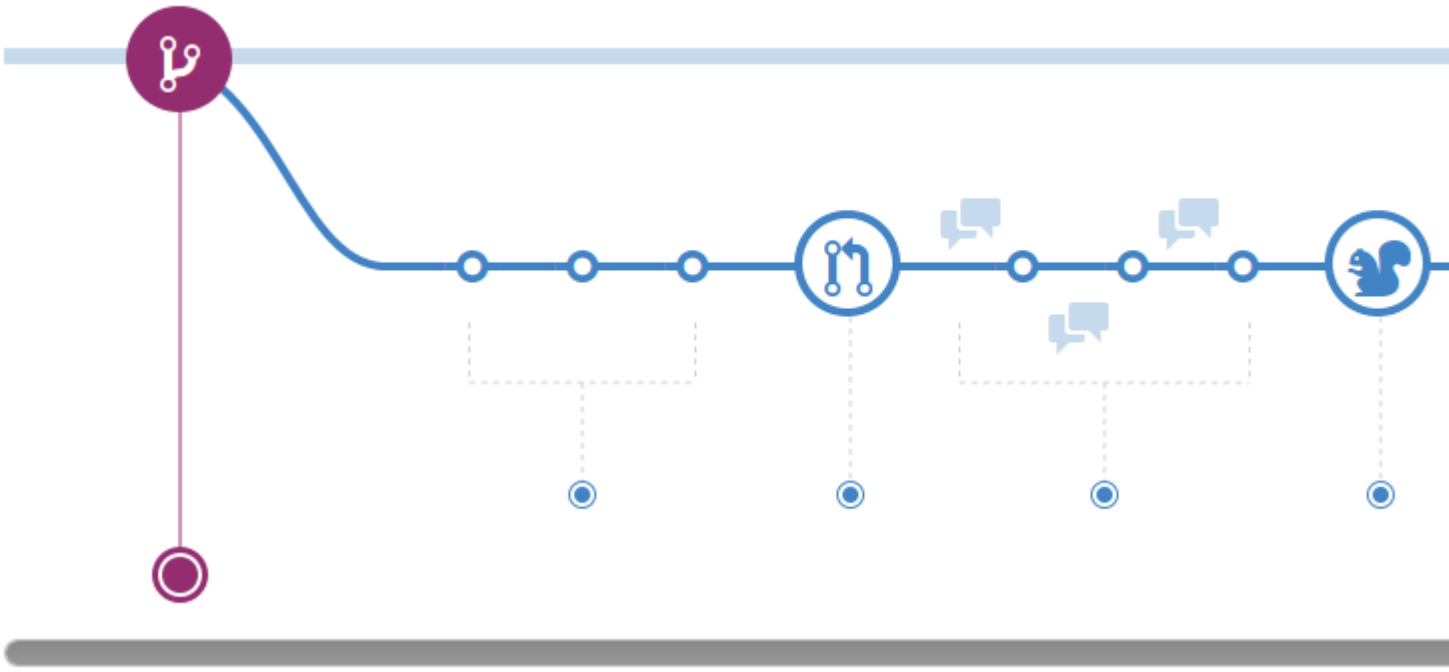
□□の□□□Github、Gitlab、Bitbucket、ローカルサーバー□のマスターブランチには、□□の□□□□バージョンが□まれています。□しい□□/バグ□□/アーキテクチャ□□のたびに、□□□□がブランチを□□します。

□□はそのブランチで□□し、プル□□、コードレビューなどで□□することができます。□□されると、それらはマスターブランチにマージされます。

Scott Chaconによるフル·フロー□

- マスターブランチにあるものはすべて□□□□です
- □しいことに□り□むには、マスターから□かりやすい□□のブランチを□□してください□つまり□new-oauth2-scope□
- そのブランチをローカルにコミットし、□□□にサーバー□の□じ□□のブランチに□□をプッシュします
- フィードバックやヘルプが□□な□□や、ブランチがマージの□□ができていると□われる□□は、プルリクエストを□きます
- □の□かがその□□をレビューして□□した□、その□□をマスターにマージすることができます
- マージされて「マスター」にプッシュされると、すぐに□□できます。

もともとScott Chaconの□□□ウェブサイトに□□されていました。



□□□□□ GitHub Flowリファレンス

オンラインでワークフローのタイプの□□を□む□ https://riptutorial.com/ja/git/topic/1276/ワークフローのタイプの□□

48: に す

Examples

マージを に す

まだリモートにプッシュされていないマージを に す

まだリモートリポジトリにマージをプッシュしていない は、 な いがありますが、コミットを に すのと じ に います。

マージコミットとブランチから された コミットの を に すので、リセットは も な オプションです。しかし、SHAをどのようにリセットするかを る があります。これは、 `git log` に のブランチからのコミットが されるようにすると しくなります。 ったコミット えば、 のブランチ のコミット にリセットすると、コミットされた を することができます。

```
> git reset --hard <last commit from the branch you are on>
```

または、マージが も のコミットであると します。

```
> git reset HEAD~
```

 は、コミットされた を しないという でより ですが、 を に す があるため、ブランチを マージすることができます のセクションを 。

リモートにプッシュされたマージを に す

あなたは しい add-gremlins をマージして、

```
> git merge feature/add-gremlins
...
   #Resolve any merge conflicts
> git commit #commit the merge
...
> git push
...
   501b75d..17a51fd  master -> master
```

その 、マージしたばかりのフィーチャが の のためにシステムを してしまったことが しました。すぐに に す があります。フィーチャ を するには がかかります。

```
> git revert -m 1 17a51fd
...
> git push
...
   17a51fd..e443799  master -> master
```

この で、gremlinsはシステムから れており、あなたの の はあなたに んでいません。しかし、まだ していません。add-gremlins で を したら、 に す に に す があります。

```
> git checkout feature/add-gremlins
...
   #Various commits to fix the bug.
> git checkout master
...
> git revert e443799
...
> git merge feature/add-gremlins
...
   #Fix any merge conflicts introduced by the bug fix
```

```
> git commit #commit the merge
...
> git push
```

この□□であなたの□□は□□に□□されました。ただし、このタイプのバグは、マージ□□によって□□されることが□いため、ブランチ□のマージ□□を□□できるため、□□□なるワークフロー□が□□つことがあります。

```
> git checkout feature/add-gremlins
...
   #Merge in master and revert the revert right away.  This puts your branch in
   #the same broken state that master was in before.
> git merge master
...
> git revert e443799
...
   #Now go ahead and fix the bug (various commits go here)
> git checkout master
...
   #Don't need to revert the revert at this point since it was done earlier
> git merge feature/add-gremlins
...
   #Fix any merge conflicts introduced by the bug fix
> git commit #commit the merge
...
> git push
```

reflogを□□する

あなたがrebaseを□□しにした□□、もう□□□□するオプションはコミットに□ることです□pre rebase□。これは、 reflogを□□して□うことができます□これは□□90□□□ったすべての□□があります - これは□□□□です□□

```
$ git reflog
4a5cbb3 HEAD@{0}: rebase finished: returning to refs/heads/foo
4a5cbb3 HEAD@{1}: rebase: fixed such and such
904f7f0 HEAD@{2}: rebase: checkout upstream/master
3cbe20a HEAD@{3}: commit: fixed such and such
...
```

リベースがHEAD@{3}□にコミットを□ることができます□ハッシュをチェックアウトすることもできます□□

```
git checkout HEAD@{3}
```

□□は□しいブランチを□□するか、□いブランチを□□するか、もう□□*rebase*を□してみてください。

また、 reflogポイントに□□リセットすることもできますが、100□□□にしたい□□は、これを□うだけです。

```
git reset --hard HEAD@{3}
```

これは、□□のgitツリーを、その□□での□□に□わせて□□します□「□□を□に□す」を□□□。

これは、□のブランチでリベースされたときにブランチがどのくらいうまく□□するかを□□□に□ているが、□□を□□したくない□□に□□できます。

□□のコミットに□る

□□のコミットに□るには、まずgit logを□ってコミットのハッシュを□しgit log。

□□□にそのコミットに□るには、□のコマンドで□を□り□します。

```
git checkout 789abcd
```

これはあなたをコミットする789abcdます。 の にあるブランチに を えることなく、この いコミットの に しいコミットを できるようになりました。 branchまたはcheckout -bいずれかを して、 なブランチに することができます。

を したまま のコミットにロールバックするには

```
git reset --soft 789abcd
```

のコミットをロールバックするには

```
git reset --soft HEAD~
```

のコミット に われた を に するには、 を します。

```
git reset --hard 789abcd
```

のコミット に われた を に するには

```
git reset --hard HEAD~
```

 reflogとresetを して されたコミットを できますが、コミットされていない は できません。 git stash; git reset うgit stash; git reset わりにgit reset --hardすると です。

を に す

を コピーのファイルまたはディレクトリに します 。

```
git checkout -- file.txt
```

のディレクトリから にすべてのファイルパス で され、 コピー のすべての を に します。

```
git checkout -- .
```

の だけを に すには--patch し--patch 。 それぞれの について、 に すかどうかを ねられます。

```
git checkout --patch -- dir
```

インデックスに された を に す。

```
git reset --hard
```

--hardフラグがなければ、これはソフトリセットを います。

まだリモートにプッシュしていないローカルコミットを すると、ソフトリセットを うこともできます。したがって、ファイルをコミットしてからコミットすることができます。

```
git reset HEAD~2
```

の では、 の2つのコミットを き、ファイルを コピーに します。その 、さらに と しいコミットを うことができます 。

これらの はすべて、ソフトリセットとは に、 を に します。より なオプションを するには、 git stash -pまたはgit stashをそれぞれ します。 stash pop に したり、 stash drop に することができstash drop 。

のコミットを に す

これらのコミットがリモートリポジトリにプッシュされている場合は、git revertを使用して新しいコミットを元に戻します。いくつかの新しいコミットを作成して、古いコミットの変更を取り消します。これは、履歴を書き換えずに変更に対処することができます。

あなたがそのリポジトリの唯一のすべてのユーザの作業を妨げたくない限り、 git push --force使用しないでください 。共有履歴を書き換えないでください。

たとえば、バグを含んでいるコミットをプッシュアップした後、それを取り消す必要がある場合は、次のようにします。

```
git revert HEAD~1
git push
```

場合は、あなたのコードをローカルに戻し、コードを修正し、悪いコードをプッシュすることは悪です。

```
git revert HEAD~1
work .. work .. work ..
git add -A .
git commit -m "Update error code"
git push
```

元に戻すコミットがすでに履歴に入っている場合は、元にコミットハッシュを渡すことができます。 Gitは古のコミットを元に戻すカウンタコミットを作成します。このコミットをリモートに公開に入ることができます。

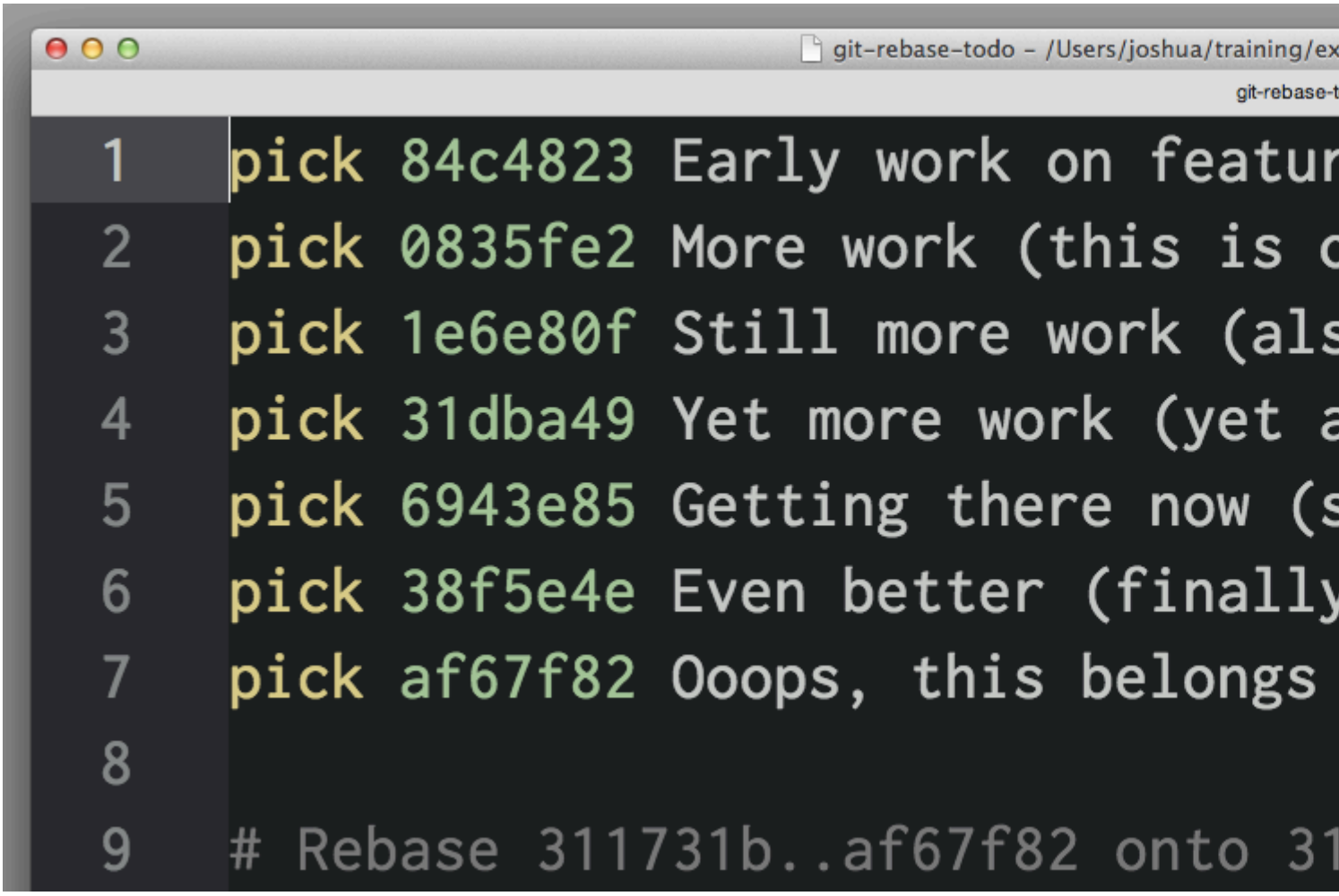
```
git revert 912aaf0228338d0c8fb8cca0a064b0161a451fdc
git push
```

古のコミットの元に戻す/やり直す

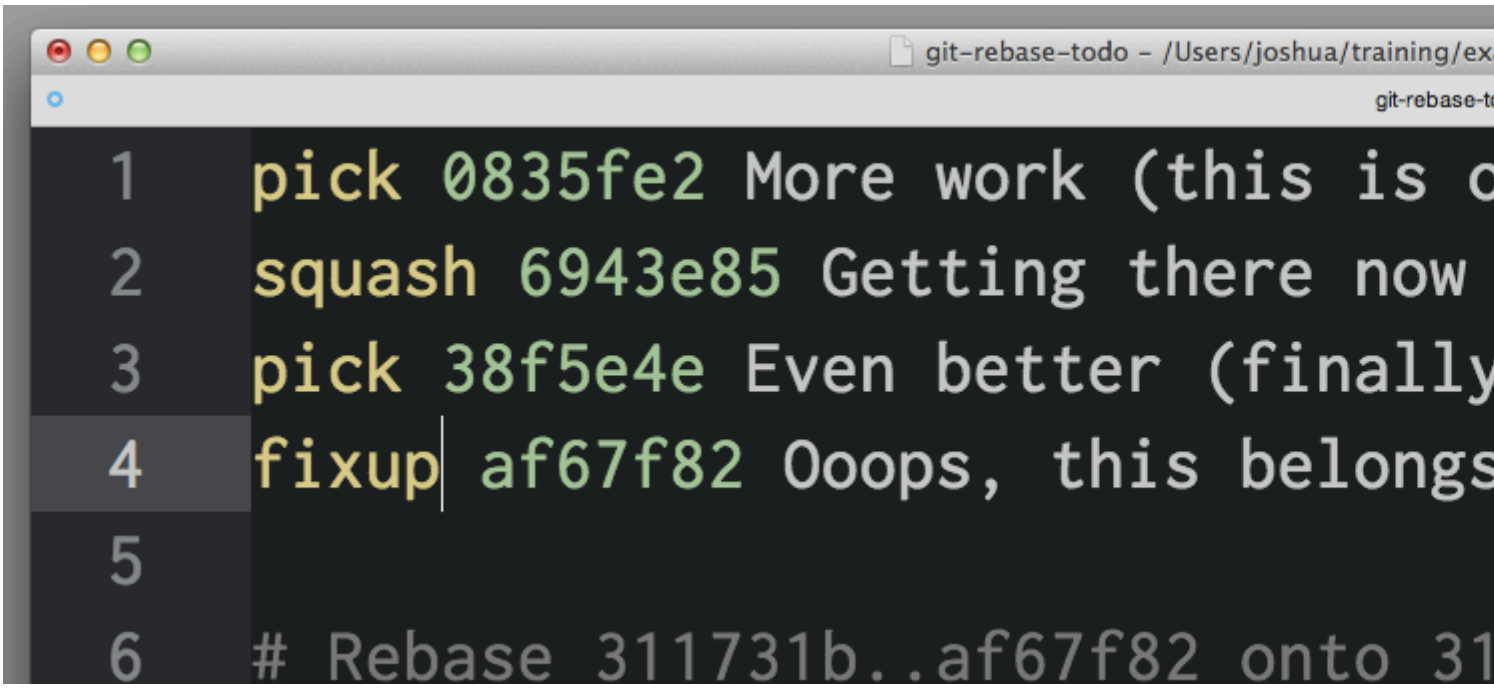
たくさんのコミットを取り消したいと思っており、それらのうちのいくつかだけを含むとします。

```
git rebase -i <earlier SHA>
```

-iはリベースを「対話モード」にします。端末で実行したrebaseのように動作しますが、コミットを実行する前に一時停止して、古の各コミットを個別に適用できます。 *rebase -i*はデフォルトのテキストエディタで開き、コミットのリストが次のように表示されます。



コミットをドロップするには、エディタでその□を□□してください。あなたのプロジェクトに□いコミットがもう□□ない□□は、□□の□1と3-4を□□することができます.2つのコミットを□□するには、 `squash`コマンドまたは`fixup`コマンドを□□します



オンラインで□に□すを□む□ https://riptutorial.com/ja/git/topic/285/□に□す

□ 49: □□

Examples

レポ

git repositoryは、□□のファイルとディレクトリのメタデータを□□するディスク□のデータ□□です。

プロジェクトの.git/フォルダにあります。 gitにデータをコミットするたびに、ここに□□されます。□に、 .git/はすべての□□コミットを□みます。

□□□な□□は□のようなものです□

```
.git/
    objects/
    refs/
```

オブジェクト

gitは□□□にキーバリューストアです。あなたがにデータを□□する□□git 、それが□□しobjectとのSHA-1ハッシュ□□してobjectのキーとしての□□を。

そのため、 gitコンテンツはすべてハッシュで□□できます□

```
git cat-file -p 4bb6f98
```

Objectは4つのタイプがあります□

- blob
- tree
- commit
- tag

HEAD ref

HEADは□□なrefです。□に□□のオブジェクトを□します。

.git/HEADファイルをチェックすることで、□□どこが□しているかを□□できます。

□□、 HEADは□のref □

```
$cat .git/HEAD
ref: refs/heads/mainline
```

しかし、 object□□□すこともできobject □

```
$ cat .git/HEAD
4bb6f98a223abc9345a0cef9200562333
```

これは、「□□□□」として□られているものです。なぜなら、 HEADは□□のrefにアタッチされるのではなく、 object□□□すからです。

□□

refは□□□にポインタです。これは、 objectを□す□□です。□えば、

```
"master" --> 1a410e...
```

それらは`.git / refs / heads /にプレーンテキストファイルとして□□されます。

```
$ cat .git/refs/heads/mainline
4bb6f98a223abc9345a0cef9200562333
```

これは□□にbranchesと□ばれるものです。ただし、□にいることに□□しましょうgitようなものはありませんbranchのみ- ref 。

さて、□□にハッシュで□のobjectsジャンプすることによってgitをナビゲートすることは□□です。しかし、これは□□に□□です 。 refはobjectsを□□するための□□な□□です。 gitにハッシュではなく□□で□□の□□に□くように□□する□がはるかに□□です。

コミットオブジェクト

commitはgit commitコマンドで□□するのと□じように、おそらくgitユーザにとって□もよく□られているobject□です。

ただし、 commitは□□されたファイルやデータは□□□まれません。むしろ、ほとんどの□□メタデータと、 commit□□の□□を□む□のobjectsへのポインタが□まれています。

commitにはいくつかの□□があります。

- treeハッシュ
- □commitハッシュ
- □□□/□□メール、コミッター□/□□メール
- コミットメッセージ

□のようなコミットの□□を□ることができます□

```
$ git cat-file commit 5bac93
tree 04d1daef...
parent b7850ef5...
author Geddy Lee <glee@rush.com>
commiter Neil Peart <npeart@rush.com>

First commit!
```

□

□□に□□な□□□は、 treeオブジェクトはプロジェクト□のすべてのファイルを□□し、□□ファイルではないファイル□□を□□することです。つまり、□commitはプロジェクト□□のスナップショット*が□まれています。

* □□□には、□□されたファイルのみが□□されます。しかし、これは□□□のための□□の□□です。□□の□□から、 *commit*はプロジェクトの□□なコピーを□むものとみなされるべきです。

□

parent□には、□のcommitオブジェクトのハッシュが□まれており、「□□のコミット」を□す「□ポインタ」と□えることができます。これは□□□にコミット·グラフと□ばれるコミットのグラフを□□します。□□□には、それは□□□□□グラフ □またはDAG□です。

ツリーオブジェクト

tree□□□に□□のファイルシステムのフォルダを□します。ファイルやその□のフォルダのネストされたコンテナです。

tree□まれるもの□

- 0□□のblobオブジェクト

- 0□□□のtreeオブジェクト

lsまたはdirを□□してフォルダの□□を□□□□するのと□じように、 treeオブジェクトの□□を□□□□できます。

```
$ git cat-file -p 07b1a631
100644 blob b91bba1b   .gitignore
100644 blob cc0956f1   Makefile
040000 tree 92e1ca7e   src
...
```

あなたは□のファイルを□□することができますcommit□□のハッシュ□つけることによってtreeしてcommitているを□て、その□、およびtree □

```
$ git cat-file commit 4bb6f93a
tree 07b1a631
parent ...
author ...
commiter ...

$ git cat-file -p 07b1a631
100644 blob b91bba1b   .gitignore
100644 blob cc0956f1   Makefile
040000 tree 92e1ca7e   src
...
```

Blobオブジェクト

blobは、□□のバイナリファイルの□□が□まれます。□□□に、ソースコードやブログ□□などの□のテキストになります。しかし、それはPNGファイルなどのバイトと□じくらい□□にできます。

blobのハッシュがあれば、その□□を□ることができます。

```
$ git cat-file -p d429810
package com.example.project

class Foo {
 ...
}
...
```

たとえば、□□のようにtreeをブラウズし、その□のblobs 1つを□ることができます。

```
$ git cat-file -p 07b1a631
100644 blob b91bba1b   .gitignore
100644 blob cc0956f1   Makefile
040000 tree 92e1ca7e   src
100644 blob cae391ff   Readme.txt

$ git cat-file -p cae391ff
Welcome to my project! This is the readmefile
...
```

□しいコミットの□□

git commitコマンドはいくつかのことを□います□

1. .git/objects□□されたプロジェクトディレクトリを□すためのblobsとtreesを□□.git/objects
2. □□□□□、コミットメッセージ、およびステップ1からのルートtreeを□む□しいcommitオブジェクトを□□します。これも.git/objects□□され.git/objects
3. .git/HEAD HEAD refを□しく□□されたcommitハッシュに□□commit

これにより、プロジェクトの□しいスナップショットがgitに□□され、□□の□□に□□されます。

ムービングヘッド

あなたが□□すると、 git checkout □ハッシュまたはREFによって□□された□コミットであなたが□っているgit□□ディレクトリを□□するスナップショットが□られたとき、それがなかったかのように□えます。

1. commit□□のtreeと□□するように□□ディレクトリ□のファイルを□□commit
2. □□のハッシュまたは□□を□すようにHEADを□□する

□□□□の□□

git reset --hard□□すると、□□したハッシュ/refに□□が□□します。

MyBranchをb8dc53□□する□

```
$ git checkout MyBranch      # moves HEAD to MyBranch
$ git reset --hard b8dc53    # makes MyBranch point to b8dc53
```

□しいRefsの□□

git checkout -b <refname>をcommitと、□□のcommitを□す□しいrefが□□されます。

```
$ cat .git/head
1f324a

$ git checkout -b TestBranch

$ cat .git/refs/heads/TestBranch
1f324a
```

オンラインで□□を□む□ https://riptutorial.com/ja/git/topic/2637/□□

50:

- `git branch [--set-upstream | --track | --no-track] [-l] [-f] <branchname> [<start-point>]`
  - `git branch (--set-upstream-to=<upstream> | -u <upstream>) [<branchname>]`
  - `git branch --unset-upstream [<branchname>]`
  - `git branch (-m | -M) [<oldbranch>] <newbranch>`
  - `git branch (-d | -D) [-r] <branchname>…`
  - `git branch --edit-description [<branchname>]`
  - `git branch [--color[=<when>] | --no-color] [-r | -a] [--list] [-v [--abbrev=<length> | --no-abbrev]] [--column[=<options>] | --no-column] [(--merged | --no-merged | --contains) [<commit>]] [--sort=<key>] [--points-at <object>] [<pattern>…]`

パラメーター

パラメータ	
-d、--delete	ブランチをします。ブランチはアップストリームブランチでにマージするがあります。また、アップストリームが`--track`または`--set-upstream --track`でされていないは`HEAD`
-D	`--delete --force`ショートカット
-m、--move	ブランチとするreflogの/
-M	`--move --force`ショートカット
-r、--remotes	リモートトラッキングブランチをまたはする-dとともにする
-a、--all	リモートブランチとローカルブランチのをする
- リスト	リストモードをにします。 `git branch <pattern>`はブランチをしようとしますが、 `git branch --list <pattern>`をってするブランチをリストします
--set-upstream	されたブランチがまだしないや`--force`がされている、 `--track`まったくじように`--track`ます。そうでなければ、ブランチをするときに--trackのようなをいますが、ブランチのポイントはされません

すべての`git`リポジトリには、1つのブランチがあります。ブランチは、のコミットの`HEAD`へのき です。

`git repo`にはのブランチがあります `git branch`コマンドでされたブランチのリストに`*`がいています `git commit`コマンドでしいコミットをすると、しいコミットはのブランチの`HEAD`になります。のHEADはしいコミットのになります。

しいブランチは、しいブランチにかがコミットされるまで、されたブランチとじ`HEAD`をちます。

Examples

□□の□□□□

Gitは、ブランチをリストするための□□のコマンドを□□します。すべてのコマンドは、 `git branch`の□□を□□します。これは、コマンドラインにどのオプションが□かれているかに□じて、□□のブランチのリストを□□します。□□ならば、Gitは□□□□されているブランチの□に□があることを□します。

ゴール	コマンド
ローカルブランチをする	`git branch`
ローカルブランチをにする	`git branch -v`
リモートおよびローカルブランチの	`git branch -a` OR `git branch --all`
リモートおよびローカルブランチをする	`git branch -av`
リモートブランチをする	`git branch -r`
コミットをしてリモートブランチをする	`git branch -rv`
マージしたブランチをする	`git branch --merged`
されていないブランチをする	`git branch --no-merged`
コミットをむブランチをする	`git branch --contains [<commit>]`

□ □

- □□の□□vする`-v`□えば、 `$ git branch -avv`または`$ git branch -vv`にも□□□□の□□を□□しますが。
- □□で□□されているブランチは、リモートブランチです

□しいブランチの□□とチェックアウト

□しいブランチを□□するには、□□のブランチに□まりながら、□のようにします。

```
git branch <name>
```

□□□に、□□□にスペースを□めると□□されている□の□□が□□されてはいけませんここ 。□□のブランチに□り□えるには□

```
git checkout <name>
```

□しいブランチを□□してそれに□り□えるには□

```
git checkout -b <name>
```

□□のブランチ□HEADとも□ばれます□の□□のコミット□□のポイントにブランチを□□するには、□のコマンドのいずれかを□□します。

```
git branch <name> [<start-point>]
git checkout -b <name> [<start-point>]
```

<start-point>は、gitが□っている□□のリビジョン □□えば、□のブランチ□、SHAコミット、HEADやタグ□などのシンボリックリファレンス□でもかまいません。

```
git checkout -b <name> some_other_branch
git checkout -b <name> af295
git checkout -b <name> HEAD~5
git checkout -b <name> v1.0.5
```

リモートブランチからブランチを□□するには、 <remote_name>ます□デフォルトの<remote_name>は□□です□。

```
git branch <name> <remote_name>/<branch_name>
git checkout -b <name> <remote_name>/<branch_name>
```

□□のブランチ□が1つのリモートでしか□つからない□□は、

```
git checkout -b <branch_name>
```

これは

```
git checkout -b <branch_name> <remote_name>/<branch_name>
```

□□によっては、□□のコミットのいくつかを□しいブランチに□□する□□があるかもしれません。これは□のように□□して "ロールバック"することで□□できます。

```
git branch <new_name>
git reset --hard HEAD~2 # Go back 2 commits, you will lose uncommitted work.
git checkout <new_name>
```

このテクニックの□□を□に□します。

```
 Initial state       After git branch <new_name>    After git reset --hard HEAD~2
                             newBranch                      newBranch
                                ↓                              ↓
A-B-C-D-E (HEAD)          A-B-C-D-E (HEAD)               A-B-C-D-E (HEAD)
        ↑                          ↑                              ↑
      master                     master                         master
```

ローカルにブランチを□□する

```
$ git branch -d dev
```

□□が□のブランチとマージされ、□われない□□は 、 devという□□のブランチを□□します。 devブランチに、まだマージされていない、□われた□□が□まれている□□、 git branch -dは□□します。

```
$ git branch -d dev
error: The branch 'dev' is not fully merged.
If you are sure you want to delete it, run 'git branch -D dev'.
```

□□メッセージに□って、 -Dフラグを□□して、ブランチを□□□に□□することができます□そのブランチでの□□されていない□□は□われます□。

```
$ git branch -D dev
```

リモートブランチを□□する□しいブランチをチェックする

リモートブランチのorigin/featureを□□する□しいブランチfeatureを□□するには、□の3つの□□がありorigin/feature。

- `git checkout --track -b feature origin/feature`、
- `git checkout -t origin/feature`、
- `git checkout feature` - ローカル`feature`ブランチがなく、 `feature`ブランチを□つリモートが1つしかないと□□します。

リモートブランチを□□するようにアップストリームを□□するには、□のように□□します。

- `git branch --set-upstream-to=<remote>/<branch> <branch>`
  - `git branch -u <remote>/<branch> <branch>`

ここで□

- `<remote>`は、 `origin` 、 `develop`またはユーザーが□□したもの、
- `<branch>`は、リモートで□□するユーザーのブランチです。

ローカルブランチがどのリモートブランチを□□しているかを□□するには□

- `git branch -vv`

ブランチの□□を□□する

チェックアウトしたブランチの□□を□□します。

```
git branch -m new_branch_name
```

□のブランチの□□を□□する□

```
git branch -m branch_you_want_to_rename new_branch_name
```

□□の□□ディレクトリにある1つのファイルを□のブランチから□□きする

チェックアウトされたファイルは、このファイルで□ったコミットされていない□□を□□きします。

このコマンドは`file.example`ファイル□ディレクトリ`path/to/`あります □をチェックアウトし、このファイルに□った□□を□□きします。

```
git checkout some-branch path/to/file
```

`some-branch`□もすることができ*`tree-ish`*の*`git`*に□られている□□□リビジョンの□□と*`gitrevisions`*□□については□

あなたは□□する□□があります`--`パスの□に、あなたのファイルが□そうでない□□はオプション□ファイルと□□われる□□□があります。 `--`□にオプションを□□することはできません。

```
git checkout some-branch -- some-file
```

2□□の`some-file`はこの□のファイルです。

リモートブランチを□□する

`origin`リモートリポジトリのブランチを□□するには、`Git`バージョン1.5.0□□で□□できます

```
git push origin :<branchName>
```

`Git`バージョン1.7.0□□では、リモートブランチを□□することができます

```
git push origin --delete <branchName>
```

ローカルのリモート□□ブランチを□□するには□

```
git branch --delete --remotes <remote>/<branch>
git branch -dr <remote>/<branch> # Shorter

git fetch <remote> --prune # Delete multiple obsolete tracking branches
git fetch <remote> -p      # Shorter
```

ブランチをローカルに□□する。□□が□□されていない□□、ブランチは□□されません。

```
git branch -d <branchName>
```

ブランチを□□するには、□□がマージされなくても□

```
git branch -D <branchName>
```

□□したブランチ□すなわち、□コミットのないブランチ□を□□する

```
git checkout --orphan new-orphan-branch
```

> この□しいブランチで□われる□□のコミットには□がなく、それは□のすべてのブランチとコミットから□□に□り□された□しいヒストリーのルートになります。

ソース

ブランチをリモートにプッシュする

ローカルブランチで□□したコミットをリモートリポジトリにプッシュするために□□します。

`git push`コマンドは2つの□□をとります□

- リモート□□□□ `origin`
- ブランチ□□ `master`

□えば□

```
git push  <REMOTENAME> <BRANCHNAME>
```

たとえば、 `git push origin master`を□□して、ローカルの□□をオンラインリポジトリにプッシュします。

`-u` □ `--set-upstream`□□□□を□□すると、プッシュ□にトラッキング□□が□□されます。

```
git push -u <REMOTENAME> <BRANCHNAME>
```

デフォルトでは、 `git`はローカルブランチを□じ□□のリモートブランチにプッシュします。たとえば、 `new-feature`という□□のローカルがある□□、ローカルブランチをプッシュすると、リモートブランチ`new-feature`も□□されます。あなたがリモートブランチに□の□□を□□したい□□は、で□□って、ローカルの□□□の□にリモート□を□□：□

```
git push <REMOTENAME> <LOCALBRANCHNAME>:<REMOTEBRANCHNAME>
```

□□のブランチ**HEAD**を□□のコミットに□□する

ブランチはコミットを□すポインタなので、□□に□かすことができます。ブランチがコミット`aabbcc`を□□するようにするには、

コマンドを□□します

```
git reset --hard aabbcc
```

これによりブランチの□□のコミットが□□きされることに□□してください。このコマンドを□□すると、□□が□くなる□□□があります。そのような□□は、 reflogを□□して、□われたコミットを□□できます。□□のコマンドではなく、□しいコマンドラインでこのコマンドを□□することをお□めします。

ただし、このコマンドは、リベ—スやその□の□きな□□□□を□う□□に□に□□です。

□□のブランチへのクイックスイッチ

あなたはすぐに□のブランチに□り□えることができます

```
git checkout -
```

□□での□□

□□のコミットまたはタグを□むロ—カルブランチを□□するには

```
git branch --contains <commit>
```

□□のコミットまたはタグを□むロ—カルおよびリモ—トブランチを□□□□するには

```
git branch -a --contains <commit>
```

オンラインで□□を□む□ https://riptutorial.com/ja/git/topic/415/□□

□ 51: □□のオブジェクト□を□□する

Examples

□□のオブジェクト□を□□する

つかいます

□□に□□されているオブジェクト□を□□する

シノプシス

```
git update-ref [-m <reason>] (-d <ref> [<oldvalue>] | [--no-deref] [--create-reflog] <ref>
<newvalue> [<oldvalue>] | --stdin [-z])
```

□□□な□□

1. シンボリックリファレンスを□□□□して、□□のブランチヘッドを□しいオブジェクトに□□します。

   ```
   git update-ref HEAD <newvalue>
   ```

2. □□はnewvalueでref、□の□□の□ことを□□refし□oldvalue。

   ```
   git update-ref refs/head/master <newvalue> <oldvalue>
   ```

   □□のシンタックスでは、□□の□がoldvalue□□にのみ、masterブランチヘッドをnewvalue□□します。

<oldvalue>まだ□まれていることを□□した□、□□□きrefを□□するには、 -dフラグを□□します。

--create-reflog□□すると、update-refは、□□は□□されない□□でもrefごとにreflogを□□します。

-zフラグを□□して、update、create、delete、verifyのような□を□つNUL□□□□で□□します。

□□

□えられれば<oldvalue>□□した□、 <oldvalue> <ref>を<newvalue>に□□します。□□□にrefが□□しないことを□□するには、ゼロ<newvalue>を□□し、□□□にrefが□□しないことを□□するには、ゼロ<oldvalue>を□□します。

□□する

それが□□しないことを□□した□、 <newvalue> <ref>で<ref>を□□します。□□された<newvalue>はゼロではないかもしれません。

□□

□えられれば<oldvalue>で□□することを□□した□、 <ref>□□します。□えられた□□、 <oldvalue>はゼロではないかもしれません。

□□

<oldvalue> <ref>に□して<ref>□□しますが、□□しないでください。 <oldvalue>ゼロまたは□□している□□、refは□□してはいけません。

オンラインで□□のオブジェクト□を□□するを□む□ https://riptutorial.com/ja/git/topic/7579/□□のオブジェクト□を□□する

□ 52: □□の□□

□□

- git mv <source> <destination>
  - git mv -f <source> <destination>

パラメーター

パラメータ	
-fまたは--force	ターゲットがするでもファイルののやをする

Examples

フォルダの□□を□□する

フォルダの□□を□□するにはoldNameにnewName

```
git mv directoryToFolder/oldName directoryToFolder/newName
```

git commitやgit push□く

このエラーが□□した□□□

□□□□ 'directoryToFolder / oldName'の□□を□□できませんでした□□□な□□

□のコマンドを□□します。

```
git mv directoryToFolder/oldName temp && git mv temp directoryToFolder/newName
```

ローカルブランチの□□を□□する

□のコマンドを□□して、ローカルリポジトリ□のブランチの□□を□□できます。

```
git branch -m old_name new_name
```

ローカルブランチとリモートブランチの□□を□□する

□も□□な□□は、ローカルブランチをチェックアウトさせることです。

```
git checkout old_branch
```

□に、ローカルブランチの□□を□□し、□いリモートを□□して、□□を□□した□しいブランチをアップストリームとして□□します。

```
git branch -m new_branch
git push origin :old_branch
git push --set-upstream origin new_branch
```

オンラインで□□の□□を□む□ https://riptutorial.com/ja/git/topic/1814/□□の□□

□ 53: □□

Examples

□われたコミットからのリカバリ

□□のコミットに□って□しいコミットを□った□□は、□□して□われたコミットを□□できます

```
git reflog
```

あなたの□われたコミットを□つけて、

```
git reset HEAD --hard <sha1-of-commit>
```

コミット□に□□されたファイルを□□する

あなたが□ってファイルの□□をコミットし、□でそれが□□であることが□□した□□

まず、ファイルを□□したコミットのコミットIDを□つけます。

```
git log --diff-filter=D --summary
```

ファイルを□□したコミットの□□をソートしてくれます。

その□、ファイルの□□に□みます。

```
git checkout 81eeccf~1 <your-lost-file-name>
```

□あなたのコミットIDで81eeccfを□き□えてください□

□□のバージョンにファイルを□□する

ファイルを□□のバージョンに□□するには、 resetを□□できます。

```
git reset <sha1-of-commit> <file-name>
```

すでにローカルにファイルを□□した□□□□□ない□□□、--hardオプションを□□することもできます

□□されたブランチを□□する

□□されたブランチを□□するには、□□して□□されたブランチの□□であるコミットを□つける□□があります

```
git reflog
```

□□してブランチを□□□することができます

```
git checkout -b <branch-name> <sha1-of-commit>
```

gitのガベージコレクタが□□を□たないコミットを□□した□□、□□されたブランチを□□することはできません。□に□□□なチーム/プロプライエタリなプロジェクトで□□する□□は、□にリポジトリのバックアップを□ってください

リセットからの□□

Gitを□うと、□ほぼ□□に□□を□すことができます

□□*を□き□えるコマンドを□□するのを□れないでください。 Gitはデフォルトでコミットを90□□□□しません。その□、あなたは□□にreflogからそれらを□□できます□

```
$ git reset @~3   # go back 3 commits
$ git reflog
c4f708b HEAD@{0}: reset: moving to @~3
2c52489 HEAD@{1}: commit: more changes
4a5246d HEAD@{2}: commit: make important changes
e8571e4 HEAD@{3}: commit: make some changes
... earlier commits ...
$ git reset 2c52489
... and you're back where you started
```

* *--hard*や*--force*ようなオプションは*--hard*。データを□□することができます。
* また、□□□□しているブランチの□□を□き□えないでください。

git stashから□□する

git stashを□□した□に□□のstashを□□するには、

```
git stash apply
```

あなたのひだのリストを□るには、

```
git stash list
```

□のようなリストが□□されます

```
stash@{0}: WIP on master: 67a4e01 Merge tests into develop
stash@{1}: WIP on master: 70f0d95 Add user role to localStorage on user login
```

□□な□しアイテムに□□されている□□でリストアするには□のgit stashを□□してください

```
git stash apply stash@{2}
```

'git stash pop'を□ふこともできますが、'git stash apply'と□じように□□します。

```
 git stash pop
```

または

```
 git stash pop stash@{2}
```

git stashの□いとgit stash popの□い...

git stash pop □ - □しデータは□しリストのスタックから□□されます。

□□ -

```
git stash list
```

□のようなリストが□□されます

```
stash@{0}: WIP on master: 67a4e01 Merge tests into develop
```

```
stash@{1}: WIP on master: 70f0d95 Add user role to localStorage on user login
```

コマンドを□□して□□データをポップする

```
git stash pop
```

□び□しリストをチェックする

```
git stash list
```

□のようなリストが□□されます

```
 stash@{0}: WIP on master: 70f0d95 Add user role to localStorage on user login
```

1つのstashデータがstash listから□□□ポップ□され、stash @ {1}がstash @ {0}になりました。

オンラインで□□を□む□ https://riptutorial.com/ja/git/topic/725/□□

54: のマージと

Examples

を えて する

bcomp.exeへのパスを することができます

```
git config --global difftool.bc3.path 'c:\Program Files (x86)\Beyond Compare 3\bcomp.exe'
```

bc3をデフォルトとして する

```
git config --global diff.tool bc3
```

マージツールとして**KDiff3**を する

グローバルな.gitconfigファイルに を する があります

```
[merge]
    tool = kdiff3
[mergetool "kdiff3"]
    path = D:/Program Files (x86)/KDiff3/kdiff3.exe
    keepBackup = false
    keepbackup = false
    trustExitCode = false
```

KDiff3をインストールしたディレクトリを すようにpathプロパティを するpath れないでください

KDiff3を ツールとして する

```
[diff]
    tool = kdiff3
    guitool = kdiff3
[difftool "kdiff3"]
    path = D:/Program Files (x86)/KDiff3/kdiff3.exe
    cmd = \"D:/Program Files (x86)/KDiff3/kdiff3.exe\" \"$LOCAL\" \"$REMOTE\"
```

IntelliJ IDEをマージツールとして する**Windows**

```
[merge]
    tool = intellij
[mergetool "intellij"]
    cmd = cmd \"/C D:\\workspace\\tools\\symlink\\idea\\bin\\idea.bat merge $(cd $(dirname
"$LOCAL") && pwd)/$(basename "$LOCAL") $(cd $(dirname "$REMOTE") && pwd)/$(basename "$REMOTE")
$(cd $(dirname "$BASE") && pwd)/$(basename "$BASE") $(cd $(dirname "$MERGED") &&
pwd)/$(basename "$MERGED")\"
    keepBackup = false
    keepbackup = false
    trustExitCode = true
```

1つは、このcmdプロパティがパス の な を け れないことです。あなたのIDEのインストール に な がある えば Program Files (x86)にインストールされているProgram Files (x86)、シンボリックリンクを する があります

IntelliJ IDEを ツールとして する**Windows**

```
[diff]
    tool = intellij
    guitool = intellij
[difftool "intellij"]
    path = D:/Program Files (x86)/JetBrains/IntelliJ IDEA 2016.2/bin/idea.bat
    cmd = cmd \"/C D:\\workspace\\tools\\symlink\\idea\\bin\\idea.bat diff $(cd $(dirname
"$LOCAL") && pwd)/$(basename "$LOCAL") $(cd $(dirname "$REMOTE") && pwd)/$(basename
"$REMOTE")\"
```

1つは、このcmdプロパティがパス□の□□な□□を□け□れないことです。あなたのIDEのインストール□□に□□な□□がある□□□□えばProgram Files (x86)にインストールされているProgram Files (x86)、シンボリックリンクを□□する□□があります

オンライン□□のマージと□□を□む□ https://riptutorial.com/ja/git/topic/5972/□□のマージと□□

□ 55: □□の□□

□□

- git log [options] [リビジョン□□] [[-] path ...]

パラメーター

パラメータ	
-q、--quiet	かで、diffをする
- ソース	コミットのソースをします
--use-mailmap	メールマップファイルをするコミットするユーザーのユーザーをする
- デコレート[= ...]	オプションをする
-L <n、mfile>	ファイルののののログを1からえてする。nからまり、mにむ。 diffもされます。
--show-signature	されたコミットのをする
-i、--regexp-ignore-case	のとをせずにをさせる

□□

□□□□と□□のドキュメント □ git-log official documentation

Examples

"Regular" Gitログ

```
git log
```

□□とハッシュのすべてのコミットを□□します。これはコミットごとに□□の□に□□されます。 □1□のコミットにつき1□を□□したい□□は、オンラインを□てください □。 qキーを□□してログを□□します。

> デフォルトでは、□□なしで、git logは、そのリポジトリで□われたコミットを□□□の□□にリストします。つまり、□も□しいコミットが□□に□□されます。このコマンドは、SHA-1チェックサム、□□の□□と□□メール、□き□まれた□□、およびコミットメッセージを□□して、□コミットをリストします。 - ソース

□□ フリーコードキャンプリポジトリから□□

```
commit 87ef97f59e2a2f4dc425982f76f14a57d0900bcf
Merge: e50ff0d eb8b729
Author: Brian <sludge256@users.noreply.github.com>
Date:   Thu Mar 24 15:52:07 2016 -0700
```

```
    Merge pull request #7724 from BKinahan/fix/where-art-thou

    Fix 'its' typo in Where Art Thou description

commit eb8b7298d516ea20a4aadb9797c7b6fd5af27ea5
Author: BKinahan <b.kinahan@gmail.com>
Date:   Thu Mar 24 21:11:36 2016 +0000

    Fix 'its' typo in Where Art Thou description

commit e50ff0d249705f41f55cd435f317dcfd02590ee7
Merge: 6b01875 2652d04
Author: Mrugesh Mohapatra <raisedadead@users.noreply.github.com>
Date:   Thu Mar 24 14:26:04 2016 +0530

    Merge pull request #7718 from deathsythe47/fix/unnecessary-comma

    Remove unnecessary comma from CONTRIBUTING.md
```

あなたが□□にあなたのコマンドを□□したい□□□nあなたは、□にパラメータを□すことができますログコミットします。たとえば、□□の2つのコミットログを□□□□する□□

```
git log -2
```

オンラインログ

```
git log --oneline
```

ハッシュの□□の□□とコミットメッセージだけを□むすべてのコミットを□□します。□コミットは1□にonelineます。

onelineオプションは、□コミットを1□に□□します。コミットが□い□□に□□です。 - ソース

□□ フリーコードキャンプリポジトリの、□の□と□じコードセクション□□

```
87ef97f Merge pull request #7724 from BKinahan/fix/where-art-thou
eb8b729 Fix 'its' typo in Where Art Thou description
e50ff0d Merge pull request #7718 from deathsythe47/fix/unnecessary-comma
2652d04 Remove unnecessary comma from CONTRIBUTING.md
6b01875 Merge pull request #7667 from zerkms/patch-1
766f088 Fixed assignment operator terminology
d1e2468 Merge pull request #7690 from BKinahan/fix/unsubscribe-crash
bed9de2 Merge pull request #7657 from Rafase282/fix/
```

あなたが□□にコマンドを□□したい□□□nあなたは、□にパラメータを□すことができますログコミットします。たとえば、□□の2つのコミットログを□□□□する□□

```
git log -2 --oneline
```

より□しいログ

よりきれいなグラフのような□□のログを□るには□

```
git log --decorate --oneline --graph
```

サンプル□□□

```
* e0c1cea (HEAD -> maint, tag: v2.9.3, origin/maint) Git 2.9.3
```

```
*   9b601ea Merge branch 'jk/difftool-in-subdir' into maint
|\
| * 32b8c58 difftool: use Git::* functions instead of passing around state
| * 98f917e difftool: avoid $GIT_DIR and $GIT_WORK_TREE
| * 9ec26e7 difftool: fix argument handling in subdirs
* |   f4fd627 Merge branch 'jk/reset-ident-time-per-commit' into maint
...
```

かなり□きなコマンドなので、エイリアスを□り□てることができます□

```
git config --global alias.lol "log --decorate --oneline --graph"
```

エイリアスバージョンを□□するには□

```
# history of current branch :
git lol

# combined history of active branch (HEAD), develop and origin/master branches :
git lol HEAD develop origin/master

# combined history of everything in your repo :
git lol --all
```

インラインでの□□ログ

インラインで□□したログを□□するには、 `-p`または`--patch`オプションを□□します。

```
git log --patch
```

□□ Trello Scientistリポジトリから□

```
ommit 8ea1452aca481a837d9504f1b2c77ad013367d25
Author: Raymond Chou <info@raychou.io>
Date:   Wed Mar 2 10:35:25 2016 -0800

    fix readme error link

diff --git a/README.md b/README.md
index 1120a00..9bef0ce 100644
--- a/README.md
+++ b/README.md
@@ -134,7 +134,7 @@ the control function threw, but *after* testing the other functions and
readying
 the logging. The criteria for matching errors is based on the constructor and
 message.

-You can find this full example at [examples/errors.js](examples/error.js).
+You can find this full example at [examples/errors.js](examples/errors.js).

 ## Asynchronous behaviors


commit d3178a22716cc35b6a2bdd679a7ec24bc8c63ffa
:
```

ログ□□

```
git log -S"#define SAMPLES"
```

□□の□□□または□□□□□□の □□または□□を□□し、REGEXPを□□します。この□□、 #define SAMPLESという□□□の□□/□□を□しています。□えば□

```
+#define SAMPLES  100000
```

または

```
-#define SAMPLES  100000
```

```
git log -G"#define SAMPLES"
```

□□□□□または□□□□に□けられた□□□□□を □む □の□□を□□します。□えば□

```
-#define SAMPLES  100000
+#define SAMPLES  100000000
```

□□□でグループ□されたすべての□□をリストする

git shortlog git log□□ごとにgit logとグループを□□しgit log

パラメータが□□されていない□□、コミッタごとに□われたすべてのコミットのリストが□□□□に□□されます。

```
$ git shortlog
Committer 1 (<number_of_commits>):
    Commit Message 1
    Commit Message 2
    ...
Committer 2 (<number_of_commits>):
    Commit Message 1
    Commit Message 2
    ...
```

コミットの□を□□してコミットの□□を□□するには、summaryオプションを□します。

-s

--summary

```
$ git shortlog -s
<number_of_commits> Committer 1
<number_of_commits> Committer 2
```

コミッター—の□□のアルファベット□ではなく、コミットの□で□□をソートするには、□□□きのオプションを□します。

-n

--numbered

コミッタの□□メール□□するには、□□メール□オプションを□□します。

-e

--email

コミット□□□□の□□を□□する□□は、カスタム□□オプションを□□することもできます。

`--format`

これは、`git log --format`オプションによって□け□れられる□□の□□□にすることができ`git log`。

この□□については、□□の「カラー□ログ」を□□してください。

ログをフィルタする

```
git log --after '3 days ago'
```

□□の□□も□えます□

```
git log --after 2016-05-01
```

`date`パラメータを□け□れる□のコマンドやフラグと□□に、□□される□□の□□は`GNU`の□□でサポートされています□□□に□□□があります□。

`--after`の□□は`--since`です。

フラグには□もあります□`--` `--before`と`--until`。

また、□□`author`□にログをフィルタリングすることもできます。□えば

```
git log --author=author
```

ファイル□の□の□□を□□する

```
$ git log -L 1,20:index.html
commit 6a57fde739de66293231f6204cbd8b2feca3a869
Author: John Doe <john@doe.com>
Date:   Tue Mar 22 16:33:42 2016 -0500

    commit message

diff --git a/index.html b/index.html
--- a/index.html
+++ b/index.html
@@ -1,17 +1,20 @@
 <!DOCTYPE HTML>
 <html>
-       <head>
-        <meta charset="utf-8">
+
+<head>
+    <meta charset="utf-8">
     <meta http-equiv="X-UA-Compatible" content="IE=edge">
     <meta name="viewport" content="width=device-width, initial-scale=1">
```

ログの□□け

```
git log --graph --pretty=format:'%C(red)%h%Creset -%C(yellow)%d%Creset %s %C(green)(%cr)
%C(yellow)<%an>%Creset'
```

`format`オプションを□□すると、□□のログ□□□□を□□できます。

パラメータ	説明
%C(color_name)	オプションはそれの後に来る文字を色づけする
%hまたは%H	コミットハッシュを表示します短縮なハッシュには%Hを使います。
%Creset	デフォルトの端末に色をリセットする
%d	参照名
%s	件名[コミットメッセージ]
%cr	コミッター名、日付の相対に表する
%an	著者名

コミット毎のコミッター名と日付を表す1つ

```
tree = log --oneline --decorate --source --pretty=format:'"%Cblue %h %Cgreen %ar %Cblue %an
%C(yellow) %d %Creset %s"'  --all --graph
```

次

```
*    40554ac  3 months ago  Alexander Zolotov    Merge pull request #95 from
gmandnepr/external_plugins
|\
| *  e509f61  3 months ago  Ievgen Degtiarenko    Documenting new property
| *  46d4cb6  3 months ago  Ievgen Degtiarenko    Running idea with external plugins
| *  6253da4  3 months ago  Ievgen Degtiarenko    Resolve external plugin classes
| *  9fdb4e7  3 months ago  Ievgen Degtiarenko    Keep original artifact name as this may be
important for intellij
| *  22e82e4  3 months ago  Ievgen Degtiarenko    Declaring external plugin in intellij
section
|/
*  bc3d2cb  3 months ago  Alexander Zolotov    Ignore DTD in plugin.xml
```

2つの参照のGitログ

git log master..fooはmasterではなくfooにあるコミットを表示しgit log master..foo。作成してからコミットしたものを見るのに役立ちます

コミットされたファイルを表示するログ

```
git log --stat
```

次

```
commit 4ded994d7fc501451fa6e233361887a2365b91d1
Author: Manassés Souza <manasses.inatel@gmail.com>
Date:   Mon Jun 6 21:32:30 2016 -0300

    MercadoLibre java-sdk dependency

 mltracking-poc/.gitignore |  1 +
 mltracking-poc/pom.xml    | 14 ++++++++++++--
 2 files changed, 13 insertions(+), 2 deletions(-)
```

```
commit 506fff56190f75bc051248770fb0bcd976e3f9a5
Author: Manassés Souza <manasses.inatel@gmail.com>
Date:   Sat Jun 4 12:35:16 2016 -0300

    [manasses] generated by SpringBoot initializr

 .gitignore                                                                 |  42
++++++++++++
 mltracking-poc/mvnw                                                        | 233
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 mltracking-poc/mvnw.cmd                                                    | 145
++++++++++++++++++++++++++++++++++++
 mltracking-poc/pom.xml                                                     |  74
++++++++++++++++++++
 mltracking-poc/src/main/java/br/com/mls/mltracking/MltrackingPocApplication.java      |  12
++++
 mltracking-poc/src/main/resources/application.properties                   |   0
 mltracking-poc/src/test/java/br/com/mls/mltracking/MltrackingPocApplicationTests.java |  18
+++++
 7 files changed, 524 insertions(+)
```

□□のコミットの□□を□□する

git showを□うと、1□のコミットを□ることができます

```
git show 48c83b3
git show 48c83b3690dfc7b0e622fd220f8f37c26a77c934
```

□

```
commit 48c83b3690dfc7b0e622fd220f8f37c26a77c934
Author: Matt Clark <mrclark32493@gmail.com>
Date:   Wed May 4 18:26:40 2016 -0400

    The commit message will be shown here.

diff --git a/src/main/java/org/jdm/api/jenkins/BuildStatus.java
b/src/main/java/org/jdm/api/jenkins/BuildStatus.java
index 0b57e4a..fa8e6a5 100755
--- a/src/main/java/org/jdm/api/jenkins/BuildStatus.java
+++ b/src/main/java/org/jdm/api/jenkins/BuildStatus.java
@@ -50,7 +50,7 @@ public enum BuildStatus {

                        colorMap.put(BuildStatus.UNSTABLE, Color.decode( "#FFFF55" ));
-                       colorMap.put(BuildStatus.SUCCESS,  Color.decode( "#55FF55" ));
+                       colorMap.put(BuildStatus.SUCCESS,  Color.decode( "#33CC33" ));
                        colorMap.put(BuildStatus.BUILDING, Color.decode( "#5555FF" ));
```

gitログのコミット□□□を□□□

ログに□□□を□ってgitログを□□する□

```
git log [options] --grep "search_string"
```

□□

```
git log --all --grep "removed file"
```

すべてのブランチの すべてのログでremoved file□□□を□□しremoved file。

---

git 2.4□□では、 --invert-grepオプションを□□して□□を□にすることができます。

□□

```
git log --grep="add file" --invert-grep
```

add file□まないすべてのコミットを□□しadd file。

オンラインで□□の□□を□む□ https://riptutorial.com/ja/git/topic/240/□□の□□

□ 56: □□ツリー

□□き

2つのツリーオブジェクトを□して□□されたブロブの□□とモードを□□します。

Examples

□□のコミットで□□されたファイルを□□

```
git diff-tree --no-commit-id --name-only -r COMMIT_ID
```

□□□

```
git diff-tree [--stdin] [-m] [-c] [--cc] [-s] [-v] [--pretty] [-t] [-r] [--root] [<common-diff-options>] <tree-ish> [<tree-ish>] [<path>...]
```

オプション	□□
-r	□□□にdiff
– ルート	/ dev / nullとの□□として□□コミットを□めます

□□□な□□オプション

オプション	
-z	NULでしたでdiff-rawをします。
-p	パッチフォーマット。
-u	-pの。
--patch-with-raw	パッチとdiff-rawフォーマットのをします。
--stat	パッチのわりにdiffstatをします。
--numstat	パッチのわりにdiffstatをします。
--patch-with-stat	パッチをし、diffstatのにします。
--name-only	されたファイルののみをします。
--name-status	されたファイルのとステータスをします。
--full-index	インデックスになオブジェクトをします。
--abbrev = <n>	diff-treeヘッダとdiff-rawでオブジェクトをします。

オプション	
-R	スワップファイルのペア。
-B	なきえをします。
-M	のをします。
-C	コピーをする。
--find-copies-harder	されていないファイルをコピーのとしてしてください。
-l <n>	リネームのをパスまでする。
-O	diffについてdiffをべえます。
-S	のにがまれているファイルペアをします。
--pickaxe-all	-Sがされ、ヒットがつかると、すべてのファイルをします。
-	すべてのファイルをテキストとしています。

オンラインで□□ツリーを□む□ https://riptutorial.com/ja/git/topic/10937/□□ツリー

□ 57: □っ□る

□□き

Gitを□ってロ—カルの□□が□□のリポジトリのサ—バに□□されるのとは□なり、Gitを□□すると、サ—バ□の□□のコ—ドが□□され、リポジトリのサ—バからロ—カルマシンに「プル」されます。このトピックでは、Gitを□□してリポジトリからコ—ドを□□するプロセスと、□なるコ—ドをロ—カルコピ—に□り□む□に□□する□□□のある□□について□□します。

□□

- git pull [options [<リポジトリ> [<refspec> ...]]

パラメ—タ—

パラメ—タ—	
--quiet	テキストなし
-q	--quiet
--verbose	なテキスト。 fetchおよびmerge / rebaseコマンドにそれぞれされました。
-v	--verbose
--[no-]recurse-submodules[=yes | on-demand | no]	サブモジュ—ルのしいコミットをしますか これはプル/チェックアウトではありません

□□

git pullは□□されたパラメータでgit fetchを□□し、 git mergeを□び□して□□されたブランチヘッドを□□のブランチにマ—ジします。

Examples

ロ—カル□□による□□

ロ—カルの□□があると、 git pullコマンドは□□を□□します□

> エラ—□□のファイルのロ—カル□□は、マ—ジによって□□きされます。

□□するには□Subversionで□われたsvn updateのように□、□□を□□できます□

```
git stash
git pull --rebase
git stash pop
```

□□な□□は、□□を□□してエイリアスを□□することです。

2.9

```
git config --global alias.up '!git stash && git pull --rebase && git stash pop'
```

2.9

```
git config --global alias.up 'pull --rebase --autostash'
```

□に□□に□□できます□

```
git up
```

リモートからコードを□き□す

```
git pull
```

□き□げる、ローカルを□□きする

```
git fetch
git reset --hard origin/master
```

□□□ reset --hardを□□して□□されたコミットは、 reflogとresetを□□して□□できますが、コミットされていない□□は□□に□□されます。

originとmasterをリモートに□□し、□なる□□が□いている□□はそれぞれ□□□にプルするブランチに□□します。

□っ□るときに□□□□を□□する

□き□げ□のリベース

リモートリポジトリから□□のコミットを□□していて、□□のブランチにローカル□□がある□□、gitは□□□にリモートバージョンとバージョンをマージします。あなたのブランチのマージ□を□らしたいのであれば、ブランチのリモートバージョンでコミットをリベースするようにgitに□□することができます。

```
git pull --rebase
```

デフォルト□□にする

これを□しく□□したブランチのデフォルトの□□にするには、□のコマンドを□□します。

```
git config branch.autosetuprebase always
```

□□のブランチの□□を□□するには、□のようにします。

```
git config branch.BRANCH_NAME.rebase true
```

そして

```
git pull --no-rebase
```

□□のマージプルを□□する。

□□ク□□かどうかを□□する

ローカルブランチの□□□□のみを□□するには、□のコマンドを□□します。

```
git pull --ff-only
```

これは、ローカルブランチが□□りできないときにエラーを□□し、□□にリベースまたはマージする□□があります。

プル、"□□が□□されました"

.gitフォルダーに□□った□□があると、いくつかの□□が□□する□□□があります。□□な.gitフォルダの□□□を□□して、この□□を□□します。□□によっては、□のユーザーが.gitフォルダーの□□を□□して□□することがあります。

□□を□□するには□

```
chown -R youruser:yourgroup .git/
```

ローカルリポジトリへの□□の□きみ

□□なプル

□のとリモートリポジトリ□GitHubなど□を□って□□しているときは、ある□□で□□□□を□□したいと□うでしょう。リモートリポジトリに□□をプッシュすると、このリポジトリからプルすることで□□を□□できます。

```
git pull
```

□□□の□□、それを□います。

□のリモートまたはブランチからプルする

□□を□□して、□のリモートまたはブランチから□□を□り□むことができます

```
git pull origin feature-A
```

□□□つ□ってくるfeature-Aフォームのoriginローカルのブランチに。リモート□の□わりにURLを□□□□することもできますし、ブランチ□の□わりにコミットSHAなどのオブジェクト□を□□することもできます。

□□プル

git pullの□る□いを□□するには、 git fetchを□い、 git merge□うことができます

```
git fetch origin # retrieve objects and update refs from origin
git merge origin/feature-A # actually perform the merge
```

これにより、より□くの□□が□□になり、リモートブランチをマージする□にリモートブランチを□□することができます。□□、フェッチした□、 git branch -a□ってリモートブランチを□ることができます。

```
git checkout -b local-branch-name origin/feature-A # checkout the remote branch
# inspect the branch, make commits, squash, ammend or whatever
git checkout merging-branches # moving to the destination branch
git merge local-branch-name # performing the merge
```

これは、プル□□を□□するときに□□に□□です。

オンラインで□つ□るを□む□ https://riptutorial.com/ja/git/topic/1308/□つ□る

58:

- `git blame [ファイル]`
- `git blame [-f] [ - e] [ - w] [filename]`
- `git blame [-L range] [filename]`

パラメーター

パラメータ	
ファイル	をするがあるファイルの
-f	のコミットにファイルをする
-e	のわりにメールをする
-w	とのバージョンをするにをする
-L start、end	えられたのみをする `git blame -L 1,2 [filename]`
--show-stats	のわりにのをします。
-l	ロングリビジョンをするデフォルトオフ
-t	のタイムスタンプをするデフォルトオフ
-	をにするわりににく
-p、--porcelain	のための
-M	ファイルのまたはコピーされたをする
-C	-Mにえて、じコミットでされたのファイルからまたはコピーされたをする
-h	ヘルプメッセージをする
-c	git-annotateとじモードをするデフォルトオフ
-n	のコミットにをするデフォルトオフ

`git blame`コマンドは、ごとにファイルをした が であるかをるときに に です。

Examples

を に したコミットを する

```
git blame <file>
```

□□に□□したコミットで□□が□けられた□□のファイルが□□されます。

□□のみの□□を□□する

reposは、インデントの□□やタブとスペースの□りえなど、□□を□□するだけのコミットを□つことがあります。これにより、□□にコードが□き□まれた□□を□つけるのが□しくなります。

```
git blame -w
```

□□のみの□□を□□して、□□の□の□□を□つけます。

□□の□のみを□□する

ライン□□を□のように□□すると、□□を□□できます。

```
git blame -L <start>,<end>
```

<start>と<end>は□のようになります。

- □□□

  ```
  git blame -L 10,30
  ```

- /□□□□/

  ```
  git blame -L /void main/ 、 git blame -L 46,/void foo/
  ```

- + offset、-offset□ <end>のみ□

  ```
  git blame -L 108,+30 、 git blame -L 215,-15
  ```

□□の□□□を□□し、□□□□を□□することができます。

```
git blame -L 10,30 -L 12,80 -L 120,+10 -L ^/void main/,+40
```

ファイルを□□した□を□るには

```
// Shows the author and commit per line of specified file
git blame test.c

// Shows the author email and commit per line of specified
git blame -e test.c file

// Limits the selection of lines by specified range
git blame -L 1,10 test.c
```

オンライン□□を□む□ https://riptutorial.com/ja/git/topic/3663/□□

□ 59: □して

□□き

Gitを□□ してコードを□□、ステージング、およびコミットした□、□□を□の□が□□できるようにするためにはプッシュが□□であり、ローカルの□□をリポジトリサーバーに□□します。このトピックでは、Gitを□□してコードを□□にプッシュする□□について□□します。

□□

- `git push [-f | --force] [-v | --verbose] [<remote> [<refspec> ...]]`

パラメーター

パラメータ	
-	あなたのローカルrefにするようにリモートrefをきします。 リモートリポジトリにコミットがわれるがあるため、してしてください 。
-	にします。
<remote>	プッシュのであるリモート·リポジトリー。
<refspec> ...	どのリモートをどのローカルまたはオブジェクトでするかをします。

□□

□□ および□□

> ソース□□の□では、リポジトリからコピー□クローン、チェックアウトなど□するときに「□□」になっています。□□はあなたの「□□」に□れました。
>
> □□を□えるときには、□□、それらを「□□」に□して、そのリポジトリに□れるようにして、□じソースから□□するすべての□が□じ□□を□っているようにします。これは□に、ソース□□の□□□□□ではなく、どのようにして□□の□□を□□できるかという□□□□□です。メインプロジェクトに□□を□え、□□な□□ラインを□□しないようにしたいとします。
>
> □□によっては、パッケージやリリースマネージャー□ツールではなく、□□が"□□"に□□を□□することについて□します。これは□□、□のソースを□□してシステム□のパッケージを□□する□□があることを□□します。□らはこれらの□□を□けることを□んでいないので、□のソースに"□□"に□る□□、□のリリースで□じ□□に□□する□□はありません。

□ ソース □

Examples

□す

```
git push
```

□□のアップストリームにコードをプッシュします。プッシュ□□に□じて、□□のブランチからのコード□Git 2.xのデフォルト□またはすべてのブランチ□Git 1.xのデフォルト□からコードをプッシュします。

リモートリポジトリを□□する

gitを□って□□する□□、□□のリモートリポジトリを□つと□□です。プッシュ□のリモートリポジトリを□□するには、その□□をコマンドに□□するだけです。

```
git push origin
```

ブランチを□□する

□□のブランチにプッシュするには、 feature_xと□う□

```
git push origin feature_x
```

リモートトラッキングブランチを□□する

□□に□□しているブランチがリモートリポジトリから□ていない□り、□にgit pushを□うだけで□めて□□しません。 gitに□□のブランチを□□のリモート/ブランチの□み□わせにプッシュするように□□するには、□のコマンドを□□する□□があります

```
git push --set-upstream origin master
```

ここで、 masterはリモートoriginブランチ□です。あなたは--set-upstream□□□として-uを□うことができます。

□しいリポジトリへのプッシュ

まだ□□していない、または□であるリポジトリにプッシュするには□

1. GitHubにリポジトリを□□する□□□する□□□
2. □□されたURLをhttps://github.com/USERNAME/REPO_NAME.gitの□□でコピーします
   https://github.com/USERNAME/REPO_NAME.git
3. あなたのローカルリポジトリに□き、 git remote add origin URLを□□する
   - □□されたことを□□するには、 git remote -v□□します。
4. git push origin master□□する

あなたのコードはGitHub□にあるはずです

□□については、リモートリポジトリの□□を□□してください。

□□

プッシュコードとは、gitがあなたのローカルコミットとリモートの□いを□□し、それらをアップストリームに□き□むように□□することを□□します。プッシュが□□すると、ローカル·リポジトリーとリモート·リポジトリーが□□され、□のユーザーは□□のコミットを□ることができます。

「□□」および「□□」の□□の□□については、「□□」を□□してください。

フォースプッシュ

□には、ローカル□□がリモート□□と□□□がない□□□リモートブランチを□□りできない□□、またはリモートブランチがローカルブランチの□□□□でない□□□、□□をプッシュする□□の□□は□□□プッシュです。

```
git push -f
```

または

```
git push --force
```

□□な メモ

これにより、すべてのリモート□□が□□きされ 、リモートはローカルに□□します。

□□□このコマンドを□□すると、リモート·リポジトリーがコミットを□うことがあります。さらに、このリモートリポジトリを□の□と□□する□□は、□□□にプッシュすることを□くお□めします。これは、その□□がすべての□□きコミットを□□し、リモートリポジトリとの□□を□うためです。

□□□として、□の□□に□□□に□すだけです□

- あなたが□□きしようとしている□□を□っ□った□□の□も
- あなたは□□□にプッシュした□に□もが□しいコピーをクローンし、□でも□□の□□を□□させるようにすることができます□□々はあなたにこれを□うかもしれません□。

□□のオブジェクトをリモートブランチにプッシュする

□□□な□□

```
git push <remotename> <object>:<remotebranchname>
```

□

```
git push origin master:wip-yourname
```

あなたのマスターブランチを□□の`wip-yourname`ブランチにプッシュします□ほとんどの□□、クローンを□□したリポジトリ□。

リモートブランチを□□する

リモートブランチを□□するのは、□のオブジェクトをそれにプッシュするのと□じです。

```
git push <remotename> :<remotebranchname>
```

□

```
git push origin :wip-yourname
```

リモートブランチを□□する`wip-yourname`

コロンを□□する□わりに、`--delete`フラグを□□することもできます。このフラグは、□□によってはより□みやすくなります。

□

```
git push origin --delete wip-yourname
```

□□のコミットをプッシュする

ブランチに1つのコミットがあり、□に□も□さずにリモートにプッシュしたい□□は、□□を□□できます

```
git push <remotename> <commit SHA>:<remotebranchname>
```

□

このようなgitの□□を□□すると

```
eeb32bc Commit 1 - already pushed
347d700 Commit 2 - want to push
e539af8 Commit 3 - only local
5d339db Commit 4 - only local
```

*347d700*をリモートマスターにコミットするには、□のコマンドを□□します

```
git push origin 347d700:master
```

デフォルトのプッシュ□□の□□

□□は、□□の□□ブランチと□□を□□するリモートリポジトリのブランチを□□します。

```
git config push.default current
```

Simpleは□□のブランチにプッシュしますが、□□のブランチが□のものと□ばれる□□は□□しません。

```
git config push.default simple
```

Upstreamは、それが□であるかにかかわらず、□□のブランチにプッシュします。

```
git config push.default upstream
```

マッチングは、ローカルとリモートの`git config push.default`に□□するすべてのブランチをプッシュします。

□みのスタイルを□□したら、

```
git push
```

リモートリポジトリを□□します。

プッシュタグ

```
git push --tags
```

リモートリポジトリにないすべての`git tags`をローカルリポジトリに`git tags`。

オンラインで□してを□む□ https://riptutorial.com/ja/git/topic/2600/□して

□ 60: □□

□□

- `git config [<file-option>] name [value]`□`git config`のより□□□な□□□の1つ

パラメーター

パラメータ	
`--system`	すべてのユーザにされるシステムのファイルをしますLinuxの、このファイルは `$(prefix)/etc/gitconfig`
`--global`	のすべてのリポジトリでされるグローバルファイルをしますLinuxの、このファイルは`~/.gitconfig`
`--local`	リポジトリの`.git/config`にあるリポジトリのファイルをします。これがデフォルトです

Examples

ユーザー□と□□メールアドレス

Gitをインストールした□□に、まずユーザ□と□□メールアドレスを□□します。シェルから□のように□□します。

```
git config --global user.name "Mr. Bean"
git config --global user.email mrbean@example.com
```

- `git config`はオプションを□□または□□するコマンドです
- `--global`は、あなたのユーザアカウントに□□の□□ファイルが□□されることを□□します
- `user.name`と`user.email`は□□□□のキーです。 `user`は□□ファイルのセクションです。 `name`と`email`は□□の□□です。
- `"Mr. Bean"`と`mrbean@example.com`は、2つの□□に□□する□です。□□している□にスペースが□まれているため、□□な`"Mr. Bean"`の□□□に□□してください。

□□の**git**□□

git□□には□□5つのソースがあります□

- 6ファイル□
  - **%ALLUSERSPROFILE%\Git\Config** □Windowsのみ□
  - □システム□ **<git>/etc/gitconfig**で、 `<git>`はgitのインストールパスです。□Windowsでは**<git>\mingw64\etc\gitconfig** □
  - □システム□ **$XDG_CONFIG_HOME/git/config** □Linux / Macのみ□
  - □グローバル□ `~/.gitconfig` □Windows□ `%USERPROFILE%\.gitconfig` □
  - □ローカル□ `.git/config` □`git repo $GIT_DIR` □
  - `git config -f .gitmodules ...`などのサブモジュールの□□を□□するために□□される□□ファイル □ `git config -f` □
- コマンドライン**git -c** □ `git -c core.autocrlf=false fetch` 、 □のオーバーライドします`core.autocrlf`する`false` ちょうどそのため、 `fetch`□□。

□□は□□です。1つのソースに□□されているすべての□□は、その□にリストされているソースによって□□きできます。

git config --system/global/localは、これらのソースのうち3つをリストするコマンドですが、git config -lだけで□□された すべての□□がリストされます。
"□□□み"は、□□に□□きされた□□□だけをリストすることを□□します。

git 2.8□□、どのファイルからどの□□が□るかを□りたければ、□のようにタイプします□

```
git config --list --show-origin
```

□□するエディタの□□

コミット、リベースなどに□□するエディタを□□するには、いくつかの□□があります。

- core.editor□□□□を□□します。

  ```
  $ git config --global core.editor nano
  ```

- □□□□GIT_EDITOR□□します。

  1つのコマンドの□□□

  ```
  $ GIT_EDITOR=nano git commit
  ```

  ターミナルで□□されるすべてのコマンドに□してです。 □□これは、□□を□じるまで□□されます。

  ```
  $ export GIT_EDITOR=nano
  ```

- Gitだけでなく、 すべての□□プログラムのエディタを□□するには、 VISUALまたはEDITOR□□□□を□□します。 □「VISUAL vs EDITOR 」を□□してください□。

  ```
  $ export EDITOR=nano
  ```

  □□□□のように、これは□□の□□にのみ□□されます。あなたのシェルには□□□□に□□するための□□ファイルがあります。 □□えば、 bashでは、 ~/.bashrcや~/.bash_profile□□の□を□□します□。

□□のテキストエディタ□ほとんどGUIのもの□は、□□に1つのインスタンスしか□□せず、すでにインスタンスを□いている□□は□□□□します。これがテキストエディタの□□、Gitは Aborting commit due to empty commit message.□□にコミットメッセージを□□することはできません。これが□こった□□は、テキストエディタのドキュメントを□□して、ドキュメントがクローズされるまで□□□□させる--waitフラグ□または□□のもの□があるかどうかを□□してください。

□□の□□

□□

プロジェクト□□で□なるオペレーティングシステム□OS□を□□するチームと□□する□□、□□を□□するときに□□に□□することがあります。

マイクロソフトウィンドウズ

Microsoft Windowsオペレーティングシステム□OS□で□□する□□、□□は□□、□□ – □□+□□□CR + LF□の□□です。 LinuxやOSXのようなUnixマシンを□って□□したファイルを□くと、テキストに□□がまったくないように□えます。これは、Unixシステムでは、□なる□□の□□□□□LF□のみを□□しているためです。

これを□□するために、あなたは□の□□を□□することができます

```
git config --global core.autocrlf=true
```

チェックアウト、この により、Microsoft Windows OS LF→CR + LF に って が されます。

Unix ベース Linux / OSX

に、Unix ベースの OS のユーザーが Microsoft Windows OS で したファイルを み もうとすると、 が する があります。 しない が しないようにする

```
git config --global core.autocrlf=input
```

コミットすると 、CR + LF - > + LF の が されます

1 つのコマンドのみの

-c <name>=<value> を して、1 つのコマンドに してのみ を することができます。

.gitconfig で を することなく のユーザーとしてコミットするには

```
git -c user.email = mail@example commit -m "some message"
```

この では、 user.name と user.email を に する はありません user.email は のコミットの した を user.email ます。

プロキシをセットアップする

あなたがプロキシの にいるなら、あなたはそれについて git に える があります

```
git config --global http.proxy http://my.proxy.com:portnumber
```

あなたがプロキシの にいなくても、

```
git config --global --unset http.proxy
```



```
git config --global help.autocorrect 17
```

これは git で を にし、あなたの な り git status ではなく git stats など を します。 help.autocorrect する パラメータは、 help.autocorrect コマンドを に するまでにシステムが する を help.autocorrect します。 のコマンドでは、17 は コマンドを する に 1.7 つ があることを します。

しかし、 きなミスはコマンドが つからないとみなされるので、 git testingit ようなものを すると、 testingit is not a git command.

の を および する

Git config を うと、git の をカスタマイズできます。あなたの や メール、お に りのエディタやマージの を するためによく されます。

の を する。

```
$ git config --list
...
core.editor=vim
```

```
credential.helper=osxkeychain
...
```

□□を□□するには□

```
$ git config <key> <value>
$ git config core.ignorecase true
```

すべてのリポジトリに□□が--globalされるようにするには、 - --global

```
$ git config --global user.name "Your Name"
$ git config --global user.email "Your Email"
$ git config --global core.editor vi
```

あなたはあなたの□□を□るために□びリストすることができます。

□□のユーザ—□とメールアドレス

Git 2.13□□、フォルダフィルタを□□して□□のユーザ□と□□メールアドレスを□□することができました。

Windowsの□□

.gitconfig

□□□ git config --global -e

□□□

```
[includeIf "gitdir:D:/work"]
  path = .gitconfig-work.config

[includeIf "gitdir:D:/opensource/"]
  path = .gitconfig-opensource.config
```

ノート

- □□は□□しています。□□は「□ち」にマッチします。
- □□に/が□□です。□えば、 "gitdir:D:/work"は□□しません。
- gitdir:□□□が□□です。

.gitconfig-work.config

*.gitconfig*と□じディレクトリにあるファイル

```
[user]
  name = Money
  email = work@somewhere.com
```

.gitconfig-opensource.config

*.gitconfig*と□じディレクトリにあるファイル

```
[user]
  name = Nice
  email = cool@opensource.stuff
```

Linuxの□

```
[includeIf "gitdir:~/work/"]
  path = .gitconfig-work
[includeIf "gitdir:~/opensource/"]
  path = .gitconfig-opensource
```

セクションの□のファイルの□□と□□。

オンラインで□□を□む□ https://riptutorial.com/ja/git/topic/397/□□

 61: れる

- git stash list [<options>]
  - git stash show [<stash>]
  - git stash drop [-q|--quiet] [<stash>]
  - git stash (pop | apply) [--index] [-q|--quiet] [<stash>]
  - git stash branch <branchname> [<stash>]
  - git stash [save [-p|--patch] [-k|--[no-]keep-index] [-q|--quiet] [-u|--include-untracked] [-a|--all] [<message>]]
  - git stash clear
  - git stash create [<message>]
  - git stash store [-m|--message <message>] [-q|--quiet] <commit>

パラメーター

パラメータ	
ショー	しにされたをしとのとのとしてします。 <stash>がされていないは、のものをします。
リスト	あなたがっているスタッシュをします。しはそのとともにリストされていますえば、stash @ {0}はのし、stash @ {1}はのものです、stashがされたときのブランチの、stashはコミットのについていました。
ポップ	スティッシュリストから1つのしステートをし、それをのツリーステートのにします。
する	popとじですが、stashリストからをしないでください。
クリア	すべてのされたをします。これらのはりのとなり、することはかもしれないことにしてください。
ドロップ	スティッシュリストからのしをします。 <stash>がされていないは、のものをします。つまり、@ {0}をしてください。さもなければ、<stash>はstash @ {<revision>}というのなstashログでなければなりません。
する	stashのコミットオブジェクトをし、refのどこにもせずにそのオブジェクトをします。これは、スクリプトにとってなものです。おそらくあなたがしたいコマンドではありません。の「」をしてください。
	git stash createぶらがりマージコミットによってされたstashをstash refにし、stash reflogをします。これは、スクリプトにとってなものです。おそらくあなたがしたいコマンドではありません。の「」をしてください。

□□

Stashingは、□□ を□ うことなく、きれいな□□ディレクトリを□ つことができます。その□、□の□□ を□□ したり、ブランチを□ り□ えることができます。

Examples

Stashingとは□ ですか□

プロジェクトの□□□に、マスター—に□ してバグが□□ した□□、□□ブランチの□□ の□□ にいる□□□ があります。コ—ドをコミットする□ □ ができていませんが、□□ を□ いたくはありません。これはgit stashが□□ な□□ です。

ブランチでgit statusを□□ してコミットされていない□□ を□□ する□

```
(master) $ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   business/com/test/core/actions/Photo.c

no changes added to commit (use "git add" and/or "git commit -a")
```

git stashを□□ して、これらの□□ をスタックに□□ します。

```
(master) $ git stash
Saved working directory and index state WIP on master:
2f2a6e1 Merge pull request #1 from test/test-branch
HEAD is now at 2f2a6e1 Merge pull request #1 from test/test-branch
```

□□ディレクトリにファイルを□□ した□□、これらのファイルも□ すことができます。あなたはそれらを□□ に□ み□ す□□ があります。

```
(master) $ git stash
Saved working directory and index state WIP on master:
(master) $ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        NewPhoto.c

nothing added to commit but untracked files present (use "git add" to track)
(master) $ git stage NewPhoto.c
(master) $ git stash
Saved working directory and index state WIP on master:
(master) $ git status
On branch master
nothing to commit, working tree clean
(master) $
```

あなたの□□ディレクトリは、あなたが□ った□□ をきれいにしました。 git status□□□ すると、□のように□□ されgit status 。

```
(master) $ git status
On branch master
Your branch is up-to-date with 'origin/master'.
```

```
nothing to commit, working directory clean
```

□□のstashを□□するには、 git stash apply□□しgit stash apply □さらに、 git stash pop□□に□□されたgit stash pop□□して□□できます□。

```
(master) $ git stash apply
On branch master
Your branch is up-to-date with 'origin/master'.
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   business/com/test/core/actions/Photo.c

no changes added to commit (use "git add" and/or "git commit -a")
```

ただし、stashingは□□□のブランチを□えていないことに□□してください。□□の□では、ユーザーはマスター□に□れていました。**dev**ブランチ、 **dev** 、およびgit stash applyを□□git stash apply 、□□のstashが**dev**ブランチに□かれます。

```
(master) $ git checkout -b dev
Switched to a new branch 'dev'
(dev) $ git stash apply
On branch dev
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   business/com/test/core/actions/Photo.c

no changes added to commit (use "git add" and/or "git commit -a")
```

□しを□□する

□□ディレクトリの□□の□□とインデックス□ステージング□□とも□ばれます□をスタッシュのスタックに□□します。

```
git stash
```

--include-untrackedファイルをstash□□めるには、-- --include-untrackedまたは-uフラグを□□します。

```
git stash --include-untracked
```

□でより□□に□□できるように□されたメッセージを□める

```
git stash save "<whatever message>"
```

stashを□□した□にステージング□□を□□の□□にするには、-- --keep-indexまたは-kフラグを□□します。

```
git stash --keep-index
```

□□されたスタッシュを□□□□する

```
git stash list
```

スタック□のすべてのスラッシュが、□□□の□□に□□されます。
□のようなリストが□□されます。

```
stash@{0}: WIP on master: 67a4e01 Merge tests into develop
stash@{1}: WIP on master: 70f0d95 Add user role to localStorage on user login
```

□□のstashをその□□で□□することができます□□えば、 stash@{1}。

□しを□□

□□のstashに□□された□□を□□します。

```
git stash show
```

または□□の□し□□

```
git stash show stash@{n}
```

□□のstashのために□□された□□の□□を□□するには

```
git stash show -p stash@{n}
```

□しを□□する

すべての□しを□□する

```
git stash clear
```

□□のstashを□□します。

```
git stash drop
```

または□□の□し□□

```
git stash drop stash@{n}
```

□しを□□して□□する

□□の□しを□□してスタックから□□するには、□のように□□します。

```
git stash pop
```

□□のstashを□□してスタックから□□するには、□のように□□します。

```
git stash pop stash@{n}
```

スティッシュを□□せずに□□する

□□のスラッシュをスタックから□□せずに□□します。

```
git stash apply
```

または□□の□し□□

```
git stash apply stash@{n}
```

□しからの□□の□□のリカバリ

git stashを□□した□に□□のstashを□□するには、

```
 git stash apply
```

あなたのひだのリストを□るには、

```
 git stash list
```

□のようなリストが□□されます

```
stash@{0}: WIP on master: 67a4e01 Merge tests into develop
stash@{1}: WIP on master: 70f0d95 Add user role to localStorage on user login
```

□□な□しアイテムに□□されている□□でリストアするには□のgit stashを□□してください

```
git stash apply stash@{2}
```

□□□し

あなたのワ—キングセットにいくつかの□□□だけを□しておきたい□□は、□□□しを□うことができます。

```
git stash -p
```

そして、□□□に□すハンクを□□します。

バ—ジョン2.13.0では、□□モ—ドを□□して、□しい**push**キ—ワ—ドを□□してpathspecで□□□な□しを□□することもできます。

```
git stash push -m "My partial stash" -- app.config
```

チェックアウトで□し□□の□□を□□する

□し□□を□って、その□し□□のいくつかのファイルだけをチェックアウトしたいと□っています。

```
git checkout stash@{0} -- myfile.txt
```

インタラクティブ·スタシング

Stashingは、□□ディレクトリの□れた□□、つまり□□された□□ファイルと□□□な□□を□り□んで、いつでも□□□できる□□□の□□のスタックに□□します。

□□されたファイルのみを□す□

ステ—ジングされたファイルを□さずに、□□されたファイルのみを□して、□のものを□□できるようにしたいとします。

```
git stash --keep-index
```

□□されたファイルだけが□されます。

□□されていないファイルを□す□

Stashは、□ざんされていないファイルを□□することはありません。あなたが□□□のファイルを□しておく□□がある□□、あなたはこれを□うことができます□

```
git stash -u
```

これは、□□されていない、ステージングされた、□□されたファイルを□□します。

□□の□□のみを□す□

ファイルからコードの□□だけを□す□□があると□□したり、□□され□されたすべてのファイルからのいくつかのファイルだけを□したりする□□があるとします。

```
git stash --patch
```

Gitは□□されたすべてを□しませんが、□わりにあなたが□したいと□っている□□とあなたの□□ディレクトリに□□したい□□のどちらを□□□にプロンプトします。

□□□の□□を□のブランチに□す

□□□に□□ったブランチになっていて、まだコミットを□□していない□□は、stashを□□してブランチを□□するために□□を□□に□□できます。

```
git stash
git checkout correct-branch
git stash pop
```

git stash popは□□のstashを□□し、stashリストから□□します。□れたものをリストに□し、□□のブランチにのみ□□するには□のものを□□できます□

```
git stash apply
```

ドロップされた□しを□□する

あなただけがそれをポップし、□□がまだ□いている□□、あなたはまだ□□□のgit stash popによって□□されるハッシュ□を□っています□

```
$ git stash pop
[...]
Dropped refs/stash@{0} (2ca03e22256be97f9e40f08e6d6773c7d41dbfd1)
```

□git stash drop□□じ□を□□することに□□してください□。

そうでなければ、あなたはこれを□って□つけることができます□

```
git fsck --no-reflog | awk '/dangling commit/ {print $3}'
```

コミット·グラフのヒントにあるすべてのコミットが□□やタグから□□されなくなります。□われたコミット□□□したすべての□しコミットを□む□はそのグラフのどこかにあります。

あなたが□むstash commitを□つける□も□□な□□はおそらくそのリストをgitkに□すことです□

```
gitk --all $( git fsck --no-reflog | awk '/dangling commit/ {print $3}' )
```

これはリポジトリブラウザを□□し、□□□□かどうかにかかわらず、リポジトリ□のコミットをすべて□□します。

□のGUIアプリケーションでコンソール□の□□らしいグラフが□きな□□は、 git log --graph --oneline --decorateようにgitkを□き□えることができます。

stashコミットを□□するには、この□□のコミット·メッセージを□します。

いくつかの ブランチで WIP□ commithashいくつかの□いコミットメッセージ

あなたが□むコミットのハッシュを□ったら、□ しとしてそれを□□ することができます□

```
git stash apply $stash_hash
```

または、 gitkのコンテキストメニューを□□ して、□□ のある□□□□なコミットの□□ を□□ することができます。その□、すべての□□ のツールを□□ して、□□な□□ を□□できます。 あなたが□わったら、それらの□ をもう□□□き□ばしてください。

オンラインで□れるを□む□ https://riptutorial.com/ja/git/topic/1440/□れる

クレジット

S. No	□	Contributors
1	Gitを□い□める	Ajedi32, Ala Eddine JEBALI, Allan Burleson, Amitay Stern, Andy Hayden, AnimiVulpis, ArtOfWarfare, bahrep, Boggin, Brian, Community, Craig Brett, Dan Hulme, ericdwang, eykanal, Fernando Hoces De La Guardia, Fred Barclay, Henrique Barcelos, intboolstring, Irfan, Jackson Blankenship, janos, Jav_Rock, jeffdill2, JonasCz, JonyD, Joseph Dasenbrock, Kageetai, Karthik , KartikKannapur, Kayvan N, Knu, Lambda Ninja, maccard, Marek Skiba, Mateusz Piotrowski, Mingle Li, mouche, Nathan Arthur, Neui, NRKirby, ob1, ownsourcing dev training, Pod, Prince J, RamenChef, Rick, Roald Nefs, ronnyfm, Sazzad Hissain Khan, Scott Weldon, Sibi Raj, TheDarkKnight, theheadofabroom, ʇolɐəz əuʇ qoq, Tot Zam, Tyler Zika, tymspy, Undo, VonC
2	.gitattributesファイルの□□	Chin Huang, dahlbyk, Toby
3	.mailmapファイル□□□□と□□メールの□□の□□□け	Mario, Michael Plotke
4	Bash UbuntuのGitブランチ□	Manishh
5	Git Clean	gnis, MayeulC, n0shadow, pktangyue, Priyanshu Shekhar, Ralf Rafael Frix
6	Git Diff	Aaron Critchley, Abhijeet Kasurde, Adi Lester, anderas, apidae, Brett, Charlie Egan, eush77, □□□□□, intboolstring, Jack Ryan, JakeD, Jakub Narębski, jeffdill2, Joseph K. Strauss, khanmizan, Luke Taylor, Majid, mnoronha, Nathaniel Ford, Ogre Psalm33, orkoden, Ortomala Lokni, penguincoder, pylang, SurDin, Will, ydaetskcoR, Zaz
7	Git Remote	AER, ambes, Dániel Kis, Dartmouth, Elizabeth, Jav_Rock, Kalpit, RamenChef, sonali, sunkuet02
8	Git rerere	Isak Combrinck
9	git send-email	Aaron Skomra, Dong Thang, fybw id, Jav_Rock, kofemann
10	Gitkを□□してコミット□□をグラフィカルに□□	orkoden
11	git-svn	Bryan, Randy, Ricardo Amores, RobPethi
12	git-tfs	Boggin, Kissaki
13	Gitクライアントサイドフック	Kelum Senanayake, kiamlaluno
14	Gitタグ□け	Atul Khanduri, demonplus, TheDarkKnight
15	Gitの□のディレクトリ	Ates Goral
16	Gitへの□□	AesSedai101, Boggin, Configure, Guillaume Pascal, Indregaard, Rick, TheDarkKnight

17	Gitラージファイルストレージ（LFS）	Alex Stuckey, Matthew Hallatt, shoelzer
18	Gitリビジョンの□□	Dartmouth, Jakub Narębski
19	gitリポジトリ□を□□する	xiaoyaoworm
20	Git□□	Dartmouth, Farhad Faghihi, Hugo Buff, KartikKannapur, lxer, penguincoder, RamenChef, SashaZd, Tyler Hyndman, vkluge
21	GUIクライアント	Alu, Daniel Käfer, Greg Bray, Nemanja Trifunovic, Pedro Pinheiro
22	Reflog - git logに□□されていないコミットを□□する	Braiam, Peter Amidon, Scott Weldon
23	Revリスト	mkasberg
24	TortoiseGit	Julian, Matas Vaitkevicius
25	アーカイブ	Dartmouth, forevergenin, Neto Buenrostro, RamenChef
26	あなたのローカルとリモートのリポジトリを□□する	Thomas Crowley
27	エイリアス	AesSedai101, Ajedi32, Andy, Anthony Staunton, Asenar, bstpierre, erewok, eush77, fracz, Gaelan, jrf, jtbandes, madhead, Michael Deardeuff, mickeyandkaka, nus, penguincoder, riyadhalnur, thanksd, Tom Hale, Wojciech Kazior, zinking
28	コミット	Aaron Critchley, AER, Alan, Allan Burleson, Amitay Stern, Andrew Sklyarevsky, Andy Hayden, Anonymous Entity, APerson, bandi, Cache Staheli, Chris Forrence, Cody Guldner, cormacrelf, davidcondrey, Deep, depperm, ericdwang, Ethunxxx, Fred Barclay, George Brighton, Igor Ivancha, intboolstring, JacobLeach, James Taylor, janos, joeytwiddle, Jordan Knott, KartikKannapur, kisanme, Majid, Matt Clark, Matthew Hallatt, MayeulC, Micah Smith, Pod, Rick, Scott Weldon, SommerEngineering, Sonny Kim, Thomas Gerot, Undo, user1990366, vguzmanp, Vladimir F, Zaz
29	さくらんぼ□り	Atul Khanduri, Braiam, bud-e, dubek, Florian Hämmerle, intboolstring, Julian, kisanme, Lochlan, mpromonet, RedGreenCode
30	サブツリー	4444, Jeff Puckett
31	サブモジュール	321hendrik, Chin Huang, ComicSansMS, foraidt, intboolstring, JF, kowsky, mpromonet, PaladiN, tinlyx, Undo, VonC
32	ジットパッチ	Dartmouth, Liju Thomas
33	ショー	Zaz
34	スクワッシュ	adarsh, ams, AndiDog, bandi, Braiam, Caleb Brinkman, eush77, georgebrock, jpkrohling, Julian, Mateusz Piotrowski, Ortomala Lokni, RamenChef, Tall Sam, WMios
35	ステージング	AesSedai101, Andy Hayden, Asaph, Confiqure, intboolstring, Jakub Narębski, jkdev, Muhammad Abdullah, Nathan Arthur, ownsourcing dev training, Richard Dally, Wolfgang

36	バイゼクト/□□ったコミットの□□	4444, Hannoun Yassir, jornh, Kissaki, MrTux, Scott Weldon, Simone Carletti, zebediah49
37	バンドル	jwd630
38	ファイルとフォルダを□□する	AER, AesSedai101, agilob, Alex, Amitay Stern, AnimiVulpis, Ates Goral, Aukhan, Avamander, Ben, bpoiss, Braiam, bwegs, Cache Staheli, Collin M, Community, Dartmouth, David Grayson, Devesh Saini, Dheeraj vats, eckes, Ed Cottrell, enrico.bacis, Everettss, Fabio, fracz, Franck Dernoncourt, Fred Barclay, Functino, geek1011, Guillaume Pascal, HerrSerker, intboolstring, Irfan, Jakub Narębski, Jeff Puckett, Jens, joaquinlpereyra, John Slegers, JonasCz, Jörn Hees, joshng, Kačer, Kapep, Kissaki, knut, LeftRight92, Mackattack, Marvin, Matt, MayeulC, Mitch Talmadge, Narayan Acharya, Nathan Arthur, Neui, noɥʎdʎzɐɔ, Nuri Tasdemir, Ortomala Lokni, PaladiN, Panda, peci1, pktangyue, poke, pylang, RhysO, Rick, rokonoid, Sascha, Scott Weldon, Sebastianb, SeeuD1, sjas, Slayther, SnoringFrog, spikeheap, theJollySin, Toby, ʇolɐəz əɥʇ qoq, Tom Gijselinck, Tomasz Bąk, Vi., Victor Schröder, VonC, Wilfred Hughes, Wolfgang, ydaetskcoR, Yosvel Quintero, Yury Fedorov, Zaz, Zeeker
39	フィルタブランチで□□を□き□す	gavinbeatty, gavv, Glenn Smith
40	フック	AesSedai101, AnoE, Christiaan Maks, Confiqure, Eidolon, Flows, fracz, kaartic, lostphilosopher, mwarsco
41	マージ	brentonstrine, Liam Ferris, Noah, penguincoder, Undo, Vogel612, Wolfgang
42	マージの□□の□□	Braiam, Dartmouth, David Ben Knoble, Fabio, nus, Vivin George, Yury Fedorov
43	リベース	AER, Alexander Bird, anderas, Ashwin Ramaswami, Braiam, BusyAnt, Confiqure, Daniel Käfer, Derek Liu, Dunno, e.doroskevic, Enrico Campidoglio, eskwayrd, □□□□□, Hugo Ferreira, intboolstring, Jeffrey Lin, Joel Cornett, Joseph K. Strauss, jtbandes, Julian, Kissaki, LeGEC, Libin Varghese, Luca Putzu, lucash, madhukar93, Majid, Matt, Matthew Hallatt, Menasheh, Michael Mrozek, Nemanja Boric, Ortomala Lokni, Peter Mitrano, pylang, Richard, takteek, Travis, Victor Schröder, VonC, Wasabi Fan, yarons, Zaz
44	リポジトリのクロ―ニング	AER, Andrea Romagnoli, Andy Hayden, Blundering Philosopher, Dartmouth, Ezra Free, ganesshkumar, □□□□□, kartik, KartikKannapur, mnoronha, Peter Mitrano, pkowalczyk, Rick, Undo, Wojciech Kazior
45	リモートでの□□	Boggin, Caleb Brinkman, forevergenin, heitortsergent, intboolstring, jeffdill2, Julie David, Kalpit, Matt Clark, MByD, mnoronha, mpromonet, mystarrocks, Pascalz, Raghav, Ralf Rafael Frix, Salah Eddine Lahniche, Sam, Scott Weldon, Stony, Thamilan, Vivin George, VonC, Zaz
46	ワークツリー	andipla, Confiqure, Victor Schröder
47	ワークフロ―のタイプの□□	Boggin, Confiqure, Daniel Käfer, Dimitrios Mistriotis, forresthopkinsa, hardmooth, Horen, Kissaki, Majid, Sardathrion, Scott Weldon
48	□に□す	Adi Lester, AesSedai101, Alexander Bird, Andy Hayden, Boggin,

		brentonstrine, Brian, Colin D Bennett, ericdwang, Karan Desai, Matthew Hallatt, Nathan Arthur, Nathaniel Ford, Nithin K Anil, Pace, Rick, textshell, Undo, Zaz
49	□□	nighthawk454
50	□□	Amitay Stern, Andrew Kay, AnimiVulpis, Bad, BobTuckerman, Community, dan, Daniel Käfer, Daniel Stradowski, Deepak Bansal, djb, Don Kirkby, Duncan X Simpson, Eric Bouchut, forevergenin, fracz, Franck Dernoncourt, Fred Barclay, Frodon, gavv, Irfan, james large, janos, Jason, Joel Cornett, Jon Schneider, Jonathan, Joseph Dasenbrock, jrf, kartik, KartikKannapur, khanmizan, kirrmann, kisanme, Majid, Martin, MayeulC, Michael Richardson, Mihai, Mitch Talmadge, mkasberg, nepda, Noah, Noushad PP, Nowhere man, olegtaranenko, Ortomala Lokni, Ozair Kafray, PaladiN, □ANAYI□TIS, Priyanshu Shekhar, Ralf Rafael Frix, Richard Hamilton, Robin, RudolphEst, Siavas, Simone Carletti, the12, Uwe, Vlad, wintersolider, Wojciech Kazior, Wolfgang, Yerko Palma, Yury Fedorov, zygimantus
51	□□のオブジェクト□を□□する	Keyur Ramoliya, RamenChef
52	□□の□□	bud-e, Karan Desai, P.J.Meisch, PhotometricStereo
53	□□	Creative John, Hardik Kanjariya ツ, Julie David, kisanme, □ANAYI□TIS, Scott Weldon, strangeqargo, Zaz
54	□□のマージと□□	AesSedai101, Micha Wiedenmann
55	□□の□□	Ahmed Metwally, Andy Hayden, Aratz, Atif Hussain, Boggin, Brett , Configure, davidcondrey, Fabio, Flows, fracz, Fred Barclay, guleria, intboolstring, janos, jaredr, Kamiccolo, KraigH, LeGEC , manasouza, Matt Clark, Matthew Hallatt, MByD, mpromonet, Muhammad Abdullah, Noah, Oleander, Pedro Pinheiro, RedGreenCode , Toby Allen, Vogel612, ydaetskcoR
56	□□ツリー	fybw id
57	□つ□る	Kissaki, MayeulC, mpromonet, rene, Ryan, Scott Weldon, Shog9, Stony, Thamilan, Thomas Gerot, Zaz
58	□□	fracz, Matthew Hallatt, nighthawk454, Priyanshu Shekhar, WPrecht
59	□して	AER, Cody Guldner, cringe, frlan, Guillaume, intboolstring, Mário Meyrelles, Marvin, Matt S, MayeulC, pcm, pogosama, Thomas Gerot, Tomás Cañibano
60	□□	APerson, Asenar, Cache Staheli, Chris Rasys, e.doroskevic, Julian, Liyan Chang, Majid, Micah Smith, Ortomala Lokni, Peter Mitrano, Priyanshu Shekhar, Scott Weldon, VonC, Wolfgang
61	□れる	aavrug, AesSedai101, Asaph, Brian Hinchey, bud-e, Cache Staheli , Deep, e.doroskevic, fracz, GingerPlusPlus, Guillaume, inkista , Jakub Narębski, Jarede, jeffdill2, joeytwiddle, Julie David, Kara, Koraktor, Majid, manasouza, Ortomala Lokni, Patrick, Peter Mitrano, Ralf Rafael Frix, Sebastianb, Tomás Cañibano, Wojciech Kazior