

<http://blog.digital-squad.net/article/123090058.html>  
<http://d.hatena.ne.jp/naokirin/20111201/1322576109>  
<http://wadahiro.hatenablog.com/entry/20101110/1289401605>  
<http://sourceforge.jp/magazine/09/02/02/0655246>  
<http://git-scm.com/book/ja/%E4%BD%BF%E3%81%84%E5%A7%8B%E3%82%81%E3%82%8B-%E6%9C%80%E5%88%9D%E3%81%AE%Git%E3%81%AE%E6%A7%8B%E6%88%90>  
<https://qiita.com/take4s5i/items/15d8648405f4e7ea3039>  
<https://camo.qiitusercontent.com/a6c07de17c4d1bfacbb8e406621fdaf60c7f484a/68747470733a2f2f71696974612d696d6167652d73746f72652e73332e616d617a6f6e6177732e636f6d2f302f37303334322f30343030313061652d306431662d613237322d393362352d3431653036656330643031302e706e67>  
<https://dackdive.hateblo.jp/entry/2016/06/06/203542>  
[http://tm.root-n.com/unix:command:git:bash\\_prompt](http://tm.root-n.com/unix:command:git:bash_prompt)  
<https://qiita.com/takayukioda/items/13c65d1b10348e79461b>

## 前提

コマンドと `tig` を使うのが良いと思う。  
`ditfftool` は `winmerge` か `meld`。

## 設定

### Windows の場合

<https://gitforwindows.org/>  
ポータブル版 `msysgit` がオススメ。  
DL して解凍。

### `.gitconfig`

local の設定

```
対象リポジトリの .git/config
```

global

```
/.gitconfig
```

### `gitconfig` のサンプル

```
[user]
name = ユーザ名
email = xxxx@aaa.bb.cc

[core]
; git status でのマルチバイトをエスケープしない
quotepath = false
; エディタ
editor = vim -c ¥"set fenc=utf-8¥"

excludesfile = /.gitignore_global
attributesfile = /.gitattributes

[color]
; diff, status, branch の表示に色を付ける
diff = auto
```

```

status = auto
branch = auto

; difftool で meld を使う
[diff]
tool = meld
[difftool "meld"]
cmd = meld $LOCAL $REMOTE
[difftool "guidiff"]
cmd = meld $LOCAL $REMOTE
[difftool "vimdirdiff"]
cmd = vim -c ¥"DirDiff $LOCAL $REMOTE¥"
[difftool "guidiff_view"]
cmd = f=$(dirdiff.sh $LOCAL $REMOTE) && sh -c ¥"meld $f && rm -rf $f¥"
[difftool "vimdirdiff_view"]
cmd = f=$(dirdiff.sh $LOCAL $REMOTE) && sh -c ¥"vim -c ¥¥¥"DirDiff $f¥¥¥" && rm -rf $f¥"
[difftool "WinMerge"]
cmd = xxx/WinMergePortable.exe -u -r ¥"$LOCAL¥" ¥"$REMOTE¥"
[difftool]
prompt = false
[diff "xlsx"]
binary = true
textconv = xlsx2csv -a

; mergetool で meld を使う
[merge]
tool = meld
ff = false
[mergetool "meld"]
cmd = meld $LOCAL $BASE $REMOTE --auto-merge --output $MERGED
[mergetool]
prompt = false
[alias]
graph = log --graph --date=short --pretty=¥"format:%C(yellow)%h %C(cyan)%ad
%C(green)%an%Creset%x09%s %C(red)%d%Creset¥"
[pull]
ff = only

```

## .gitattributes

```

*.xlsx diff=xlsx
*.XLSX diff=xlsx
[push]
autoSetupRemote = true

```

## .bashrc

git-completion を使えるようにしておく と便利。

また、プロンプトに git の状況を表示するために .bashrc に以下を追記しておく と見やすい。

```

# git
gitps1=true
GIT_CONTRIB_DIR=/usr/share/git-core/contrib
if [ -r ${GIT_CONTRIB_DIR}/completion/git-prompt.sh ]; then
. ${GIT_CONTRIB_DIR}/completion/git-prompt.sh
gitps1="__git_ps1"
fi
GIT_PS1_SHOWDIRTYSTATE=true
GIT_PS1_SHOWUNTRACKEDFILES=true
GIT_PS1_SHOWSTASHSTATE=true
GIT_PS1_SHOWUPSTREAM=auto

export PS1='[¥u@¥h ¥W$(¥{gitps1})]¥$ '
# デフォルト
# '[¥u@¥h ¥W]¥$ '

```

## 使い方

### proxy を設定する

```
git config http.proxy http://my.proxy.url:80
```

とか

```
git config http.proxy http://user:pass@my.proxy.url:80
```

恒久的に設定するには --global オプションを使って

```
git config --global http.proxy http://my.proxy.url:80
```

## リポジトリ作成

```
cd path/to/repo  
git init
```

```
git init --bare
```

とするとになります。(bare リポジトリ)

bare リポジトリは、サーバ側のリモートリポジトリとして使うリポジトリを作成する際に使用します。

## 基本的なコマンド

git init

リポジトリ作成

git --bare init

bare 作業ファイルがなく、管理ファイルのみのリポジトリ (bare リポジトリ) 作成。

git add .

現在のワークツリーを記録する。コミットはされない。

git add -u

ワークツリー内の全てのファイルを記録する。

git add -A

新しく作成されたファイルを含めてインデックスに記録する。

git add -p

どの変更を次回のコミットに含めるかを選択して、インデックスに記録する。

git commit

-m オプションを省略すると、エディタが起動し、ログメッセージを残せる。

git commit -m " 最初のコミット "

コミットする。

git commit -a -m " ログメッセージ "

gitt add -u した後すぐ git -commit のと同義。

git commit -v

git commit、git diff の結果がまとめて表示される。

git commit --amend

一つ前のコミットを破棄して新しくコミットする。

git diff

前回の git add 以降に加えた変更が確認できる。

git diff HEAD

前回の git commit 以降に加えた変更が確認できる。

git diff --cached

最新のコミットと、インデックスの違いを表示。

git status

git commit を実行した場合にどのファイルへの変更がコミットされ、どのファイルへの変更はコミットされないかと確認できる。

git log --grep= パターン

ログメッセージを文字列を元に検索。複数の --grep オプションを指定した場合は OR 検索になる。AND 検索にするには --all-match オプションを加える。

git show

最新のコミットの内容を表示。

git reset

インデックスを現在の HEAD と同じにする。

git revert

指定したコミット ID へ戻す

git checkout

ブランチを変更する

git checkout -b ブランチ名

新規にブランチを作成してそのブランチに移動する。

git branch ブランチ名

ブランチ名のブランチを作成する

git branch

ブランチを全て表示する。現在のブランチの横に \* が表示される。

git merge ブランチ名

現在のブランチと、指定したブランチをマージする。

git merge -Xignore-all-space ブランチ名

現在のブランチと、指定したブランチをスペースを無視してマージする。

git clone A.git B.git

A リポジトリをコピーする。

git fetch

リポジトリから取得

git pull

リポジトリから取得。追跡ブランチをマージ。git pull --rebase でマージではなく、rebase する

## 使い方いろいろ

ローカルの変更内容確認

```
git difftool -d
```

コミット予定の内容確認

```
git difftool -d --staged
```

または

```
git difftool -d --cached
```

## ログをツリー表示する

```
log --graph --date=short --pretty="format:%C(yellow)%h %C(cyan)%ad %C(green)%an%Creset%x09%s %C(red)%d%Creset"
```

または、.gitconfig に

```
[alias]
graph = log --graph --date=short --pretty="format:%C(yellow)%h %C(cyan)%ad %C(green)%an%Creset%x09%s %C(red)%d%Creset"
```

を設定して

```
git graph
```

## 他のブランチのコミットを適用する

```
git cherry-pick ハッシュ
```

範囲でコミットを指定するときは

```
git cherry-pick ハッシュ .. ハッシュ
```

始点となるコミットは実際に cherry-pick したいコミットの1つ前を指定する

複数のコミットを1つのコミットにまとめる

```
git rebase -i ハッシュ
```

指定するコミットはまとめるコミットの始点の1つ前をしている

コマンド	説明
(p)pick	コミットをそのまま残す。
(r)reword	コミットメッセージを変更。
(e)edit	コミット自体の内容を編集。
(s)squash	直前の pick を指定したコミットに統合。メッセージも統合。
(f)fixup	直前の pick を指定したコミットに統合。メッセージは破棄。

## 1つ前のコミットを修正する

```
git commit --amend
```

staged されているファイルがない場合はコミットメッセージを修正する。

staged されているファイルがあれば、staged されているファイルが1つ前のコミットに追加される。

### コミット前に戻す

```
git reset HEAD
git reset --head HEAD
git clean
git clean -f
```

### コミット同士の差分

```
git diff ハッシュ ハッシュ
```

### リモートの情報表示

```
git remote
```

### ブランチの追跡情報表示

```
git branch -vv
```

### リモート追跡を設定する ( 上流ブランチをセットする )

```
git branch --set-upstream-to=origin/<branch> ローカルブランチ
```

### リモート追跡をやめる ( 上流ブランチをアンセットする )

```
git branch --unset-upstream ローカルブランチ
```

### 過去のコミットからブランチを作る

```
git checkout -b branch_name 336e00890fc03ba55563998d7459771649fb46b5
```

### 指定した文字列を含む変更をログから探す

```
git log -S "hoge"
git log -p -S "hoge"
```

-p は指定した文字列を含む変更箇所を表示する

### 操作を誤った時

マージしていないブランチを消したとか reset --hard してしまった場合、

```
git reflog
```

ローカル操作の履歴が見れる。

指定したハッシュ値がどのブランチに含まれているか確認

```
git name-rev
```

でハッシュ値を human-readable な形式を返す。

未プッシュのブランチを表示

```
git push --all --dry-run
```

未プッシュのタグを表示

```
git push --tags --dry-run
```

特定のディレクトリ、ファイルに影響するコミットを表示する

```
git log -- ディレクトリ名またはファイル名
```

見やすくするのに oneline を付けるのも良い

```
git log --oneline -- ディレクトリ名またはファイル名
```

## タグ

タグをつける

```
git tag タグ名
```

タグをリモートへプッシュ

```
git push --tags
```

タグを削除

ローカル

```
git tag -d タグ
```

リモート

```
git push origin :refs/tags/ タグ名
```

行を指定して履歴を確認する

直近のログ

hoge.txt の 100 行目の直近の変更に関する情報取得

```
git blame hoge.txt -L 100,100
```

履歴を表示

hoge.txt の 100 行目の変更履歴を表示

```
git log --no-patch -L 100,100:hoge.txt
```

## Git で Excel の差分を表示する

### git diff で excel の差分を表示する

xlsx2csv をインストールする

```
pip3 install xlsx2csv
```

.gitconfig

```
[core]
  attributesfile = /.gitattributes

[diff "xlsx"]
  binary = true
  textconv = xlsx2csv -a
```

.gitattributes

```
*.xlsx diff=xlsx
*.XLSX diff=xlsx
```

### git difftool -d で xlsx の差分を表示する

\$LOCAL \$REMOTE の xlsx を CSV に変換したディレクトリを用意して、そのディレクトリを比較する

dirdiff.sh

```
#!/bin/bash

# pip install xlsx2csv

if [ $# -le 1 ]; then
  echo 引数エラー: $*
  exit 1
fi

function createDir() {
  src=$1
  dest=$2

  op=-rL
  if [ -d "$src" ]; then
    op=${op}T
  fi

  cp $op "$src" $dest
  find $dest -type f -iname "*.xlsx" | xargs -l @ sh -c 'xlsx2csv -a "@" "@.csv" && rm "@"'
  # find $dest -type f -iname "*.csv" | xargs -l f sh -c 'csvlook f > f.tmp ; cp f.tmp f && rm
f.tmp'
  # find $dest -type f -iname "*.csv" | xargs -l @ sh -c 'tabulate -o "@.csv" -1 -s , -f grid "@";
cp "@.csv" "@" && rm "@.csv"'
  find $dest -type f -iname "*.csv" | xargs -l @ sh -c 'column -t -s, "@" > "@.csv" && cp "@.csv"
@" && rm "@.csv"'
}

# cmd = t1=`mktemp` && `pandoc -t plain $LOCAL >$t1` && t2=`mktemp` && `pandoc -t plain $REMOTE >$t2
` && meld $t1 $t2 && rm -f $t1 $t
dest1=$(mktemp -d)
dest2=$(mktemp -d)

createDir "$1" $dest1
```

```
createDir "$2" $dest2
echo $dest1 $dest2
```

.gitconfig

```
[difftool "guidiff_view"]
  cmd = f=$(dirdiff.sh $LOCAL $REMOTE) && sh -c ¥"meld $f && rm -rf $f¥"
[difftool "vimdirdiff_view"]
  cmd = f=$(dirdiff.sh $LOCAL $REMOTE) && sh -c ¥"vim -c ¥¥¥"DirDiff $f¥¥¥" && rm -rf $f¥"
```

## set-upstream を自動設定する

<https://zenn.dev/ktansai/articles/b9b342bc80d868>

```
git config --global --add --bool push.autoSetupRemote true
```

## Git でよく使われるコマンドをイラストで説明

<https://qiita.com/kozy/items/b42ba59a8bac190a16ab>

[Git でよく使われるコマンドをイラストで説明](#)

## GUI ツール

### git 標準

<http://www.devlog.alt-area.org/?p=1874>

```
gitk
gitk --all
git gui
```