

Gitでバージョン管理を試みよう！

Kenta Arai

はじめに

- このスライドは，私が所属していた研究室に新たに配属された学部生向けに作ったものを，加筆・修正したものです
- 使用する頻度の高いGitの機能と関連するコンテンツに焦点を当て，簡単なCプログラムのバージョン管理をする過程でGitを習得することを目指しています
- 習得の上で困難と判断した機能や概念等に関しては簡単化の為に厳密さに欠ける説明をしています
- より正確な情報を手に入れたい場合はリファレンスマニュアルを参照して下さい

<https://git-scm.com/docs>

アウトライン

- Gitとは
- Gitの基礎
- リモートリポジトリを使う
- ブランチを使おう
- おまけ) Visual Studio CodeでGitしよう
- 更新履歴

アウトライン

- Gitとは
- Gitの基礎
- リモートリポジトリを使う
- ブランチを使おう
- おまけ) Visual Studio CodeでGitしよう
- 更新履歴

Gitとは



- 分散型のバージョン管理ツール
 - リポジトリと呼ばれるデータベースで変更履歴を記録する
 - Linus TorvaldsがLinux Kernelの開発のために作った
- Gitの良いところ
 - いつでもファイルを以前の状態に戻せる
 - オフラインで開発を進められる
 - みんなで使うデータベースと自分だけのデータベースを使い分けられる
 - 多くのソフトウェアがGitで管理されている
 - つまり使い方が分かればそのソフトウェアを自分でも開発できる

GitとGitHub

- Git : バージョン管理ツール

```
$ git --version  
git version 2.17.1
```

- GitHub : 開発プラットフォーム
 - Gitリポジトリ（要はデータベース）をインターネット上で公開するサービス
 - Issue管理, Pull Request, wiki等の機能も持つ



バージョン管理（ツール）って必要？

- 開発物は時々刻々と変更が加えられていく
 - 機能の追加や変更
 - バグの修正
- バージョン管理をしないと...
 - 昔の状態に戻れなくなる
 - さっきまでは動いたのに！というときに
どうしようもない
- 人力でバージョン管理する？
 - 各バージョンの関係を
把握することが難しくなる
 - プロジェクトが肥大化する

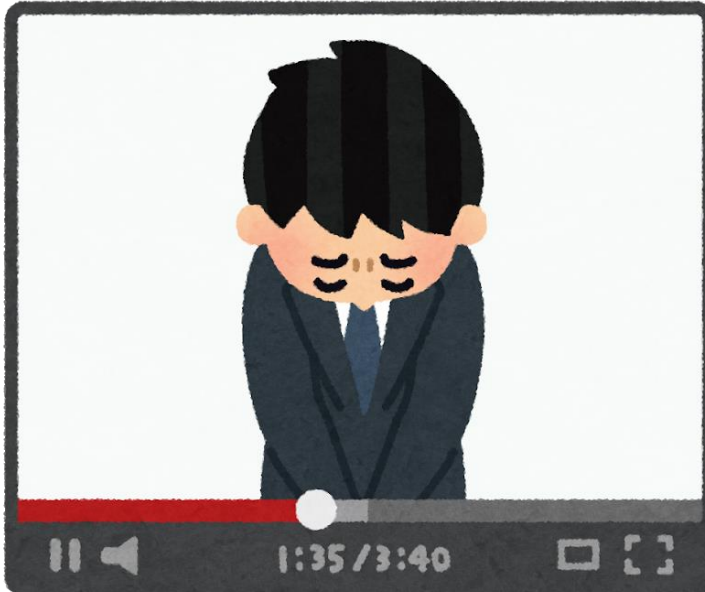


Git環境が無い人の嘆き

HIKAKINで学ぶバージョン管理の重要性(1/2)



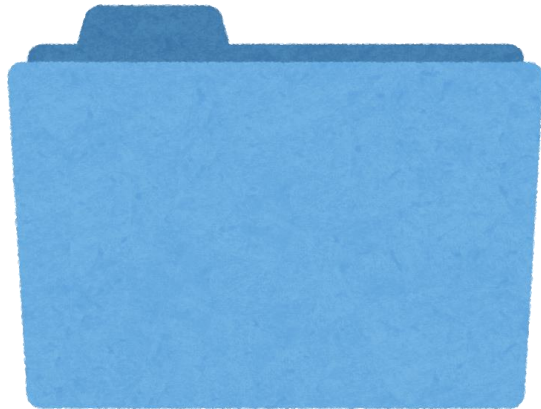
- 動画のOPとEDを新しくした
(新ヒカキンTV スタート!!,
[https://www.youtube.com/watch?v= fxIVudzl2o](https://www.youtube.com/watch?v=fxIVudzl2o))



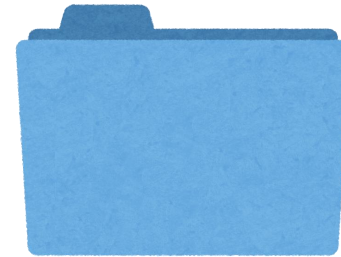
- しかし後日OPの不備が発覚...
(もふこに謝罪します...大変な
ことをしてしまいました...
【もっちゃん】 ,
https://www.youtube.com/watch?v=Tu9F_rHxgHQ&t=312s)

HIKAKINで学ぶバージョン管理の重要性(2/2)

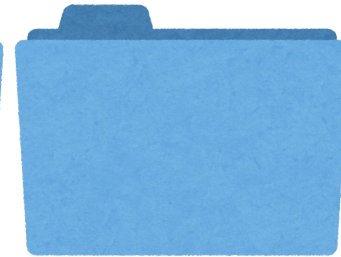
- 人力でバージョン管理をした結果、どれが本当の最終版か分からなくなってしまった...



新OP 最終



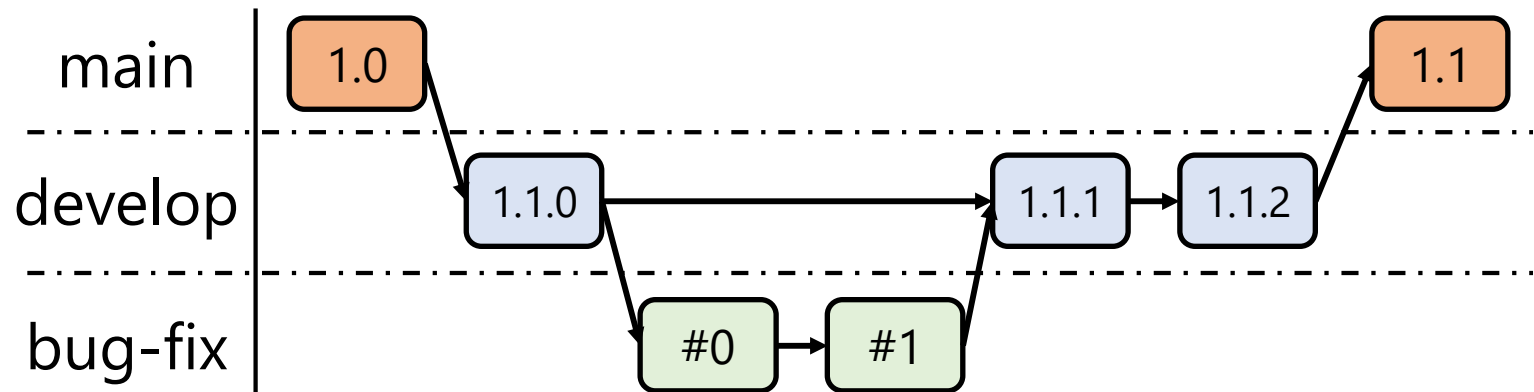
新OP 最終02



新OP 最終03

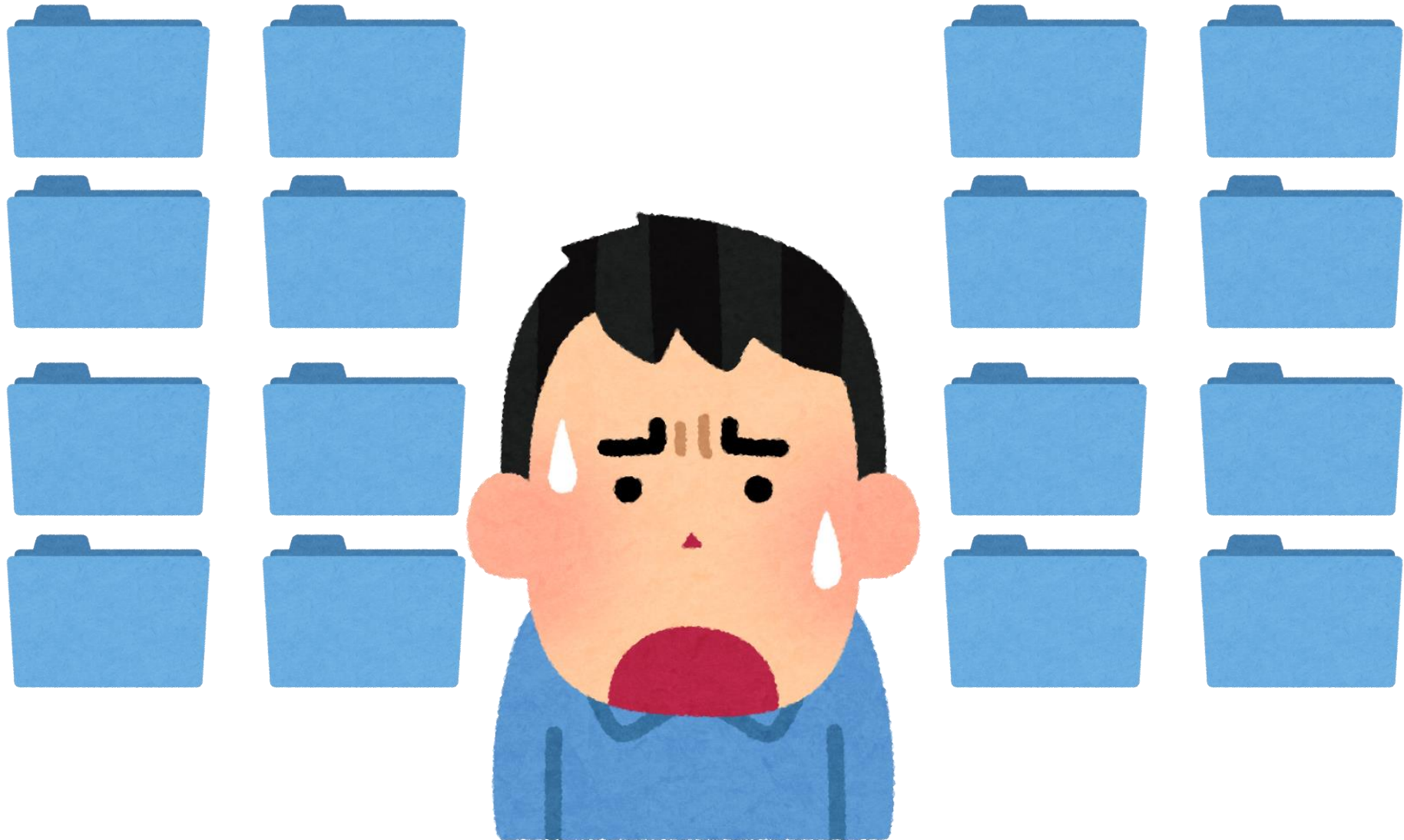
人カバージョン管理

- 良い点
 - お手軽
- 悪い点
 - 各バージョンの関係性を把握することが難しい
 - ファイル一式をzipで管理する場合は開発者が関係を把握しなければならない
 - そもそもバージョンは一本道に成長しないので複雑になる



よくあるバージョン管理の流れ

こうなる前にバージョン管理ツールを使おう



アウトライン

- Gitとは
- **Gitの基礎**
- リモートリポジトリを使う
- ブランチを使おう
- おまけ) Visual Studio CodeでGitしよう
- 更新履歴

Gitの基礎

- この章で学ぶこと
 - Gitの設定
 - config
 - リポジトリの作り方・使い方
 - init, status, add, commit, log, reset

Gitの設定

- リポジトリを使うには名前とメールアドレスが必要
 - 誰がリポジトリにファイルを登録したか明らかにするため
- configコマンド：gitの設定に使う
 - 使い方：git config <type> <設定項目> <設定内容>
- 練習：gitを使うための設定をしよう
 - ユーザ名とメールアドレスを設定しよう
 - 設定を表示させてみよう

```
2020-04-04 00:01:59 [kenta@Delphinium ~]
$ git config --global user.name "Kenta Arai"
2020-04-04 00:02:05 [kenta@Delphinium ~]
$ git config --global user.email kenta@mymail
2020-04-04 00:02:09 [kenta@Delphinium ~]
$ git config --list
user.name=Kenta Arai
user.email=kenta@mymail
```

Gitの設定は~/.gitconfigにある

- `git config --global`で設定した内容は~/.gitconfigに置かれる
- このファイルを編集して設定を変更してもよいが正しく書かないとgitが動かなくなる可能性がある

```
2020-04-04 00:02:09 [kenta@Delphinium ~]
$ git config --list
user.name=Kenta Arai
user.email=kenta@mymail
2020-04-04 00:02:11 [kenta@Delphinium ~]
$ cat .gitconfig
[user]
    name = Kenta Arai
    email = kenta@mymail
```

リポジトリを作る

- initコマンド：リポジトリを作成
 - 使い方：git init
- 練習：リポジトリを作成しよう

```
2019-05-12 16:57:39 [kenta@Delphinium git]
$ mkdir hello
2019-05-12 16:57:42 [kenta@Delphinium git]
$ cd hello
2019-05-12 16:57:45 [kenta@Delphinium hello]
$ ls -a
.  ..
2019-05-12 16:57:47 [kenta@Delphinium hello]
$ git init
Initialized empty Git repository in /home/kenta/git/hello/.git/
2019-05-12 16:57:51 [kenta@Delphinium hello]
$ ls -a
.  ..  .git <-" .git"ができる！
```


リポジトリに追加するファイルを作る

- 以下のファイルを作成

main.c

buffers

```
1 #include "hello.h"
2
3 int
4 main(int argc, char* argv[]) {
5     printHello("Kenta");
6     return 0;
7 }
```

hello.h

buffers

```
1 #ifndef __HELLO_H__
2 #define __HELLO_H__
3
4 void printHello(const char *name);
5
6 #endif // __HELLO_H__
```

hello.c

buffers

```
1 #include "hello.h"
2
3 #include <stdio.h>
4
5 void printHello(const char *name) {
6     printf("Hello, %s!\n", name);
7 }
```

Makefile

buffers

```
1 SRCS = \
2     hello.c \
3     main.c
4
5 OBJS = $(subst .c,.o,$(SRCS))
6 TARGET = hello
7
8 .SUFFIXES: .c .o
9
10 all : $(TARGET)
11
12 $(TARGET) : $(OBJS)
13     gcc -o $@ $(OBJS) $(LIBS)
14
15 .c.o :
16     gcc -c $(CFLAGS) -I. $< -o $@
17
18 clean :
19     rm -f *.o $(TARGET)
```

実はここにファイルが置いてあります

- Webにおいてあるのでコピーすること

```
2020-04-04 14:19:18 [kenta@Delphinium ~]
$ curl https://kenta11.github.io/assets/2020-04-04-git-introduction/source_code.tar.gz -o source_code.tar.gz
2020-04-04 14:19:46 [kenta@Delphinium ~]
$ tar zxvf source_code.tar.gz
source_code/
source_code/Makefile
source_code/hello.c
source_code/hello.h
source_code/main.c
2020-04-04 14:19:51 [kenta@Delphinium ~]
$ ls source_code
Makefile  hello.c  hello.h  main.c
```

プログラムの動作チェック

- makeコマンドでビルドする
- 確認が終わったら`make clean`で生成物を削除

```
2019-05-12 17:09:57 [kenta@Delphinium hello]
$ ls
Makefile hello.c hello.h main.c
2019-05-12 17:09:58 [kenta@Delphinium hello]
$ make
gcc -c -I. hello.c -o hello.o
gcc -c -I. main.c -o main.o
gcc -o hello hello.o main.o
2019-05-12 17:10:00 [kenta@Delphinium hello]
$ ls
Makefile hello hello.c hello.h hello.o main.c main.o
2019-05-12 17:10:02 [kenta@Delphinium hello]
$ ./hello
Hello, Kenta!
2019-05-12 17:10:04 [kenta@Delphinium hello]
$ make clean
rm -f *.o hello
2019-05-12 17:11:51 [kenta@Delphinium hello]
$ ls
Makefile hello.c hello.h main.c
```

リポジトリの状態を確認する

- statusコマンド：ディレクトリ内のファイルの状態を表示する
- 練習：statusコマンドを実行してみよう

```
2019-05-12 17:13:06 [kenta@Delphinium hello]
$ git status
ブランチ master

No commits yet

追跡されていないファイル:
(use "git add <file>..." to include in what will be committed)

    Makefile
    hello.c
    hello.h
    main.c

nothing added to commit but untracked files present (use "git add" to track)
```

git statusの表示はどういう意味？

ムッ、リポジトリには無いファイルが
ワーキングツリーに追加されたぞ

表示

```
2019-05-12 17:13:06 [kenta@Delphinium hello]
$ git status
ブランチ master

No commits yet

追跡されていないファイル:
(use "git add <file>..." to include in what will be committed)

    Makefile
    hello.c
    hello.h
    main.c

nothing added to commit but untracked files present (use "git add" to track)
```

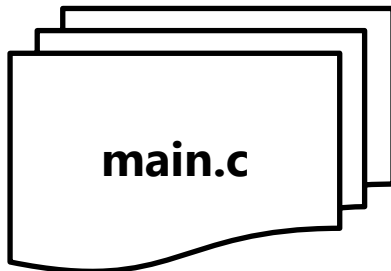
git statusの際に確認

追跡されていないファイルとして報告

ワーキングツリー

ステージングエリア

リポジトリ



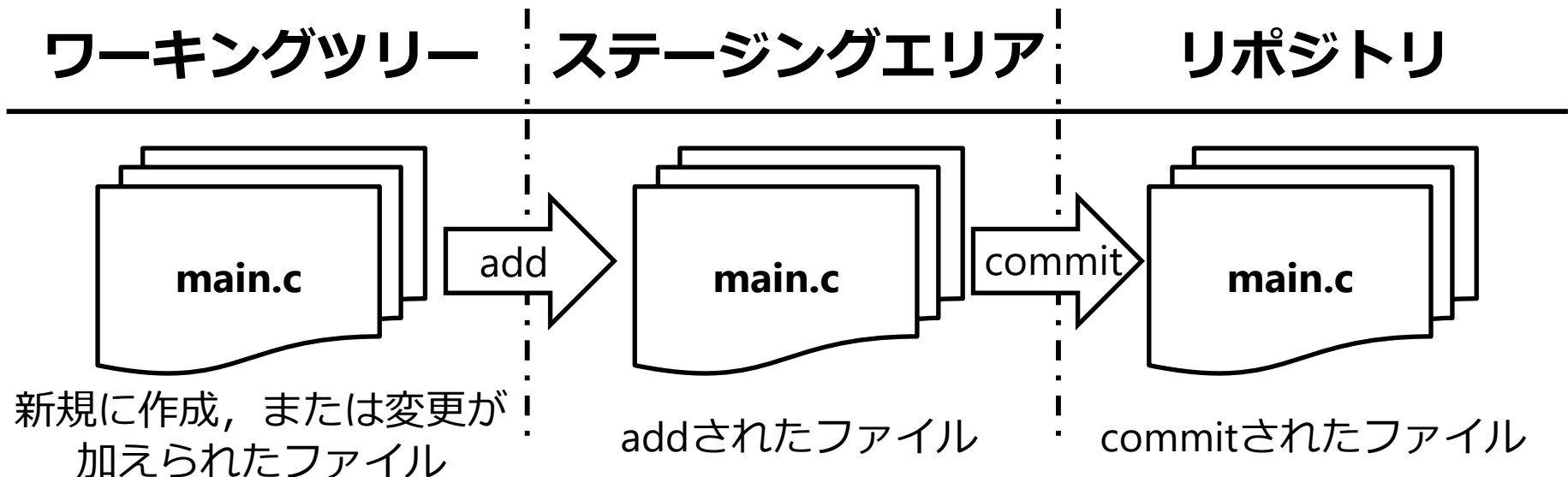
新規に作成されたファイル

何もない
(次ページで解説)

何もない

ステージングエリア

- `リポジトリに追加したいファイル`を保持しておく
 - 中間のファイル置き場だと思えば良い
- addコマンドで追加したファイルは
commitコマンドでリポジトリに記録される



ステージングエリアにファイルを追加する

- addコマンド：ステージングエリアにファイルを追加
- 使い方：git add <ファイル名>
- `git status`の表示にもaddしろと書かれている...

```
2019-05-12 17:13:06 [kenta@Delphinium hello]
$ git status
ブランチ master

No commits yet

追跡されていないファイル:
(use "git add <file>..." to include in what will be committed)

    Makefile
    hello.c
    hello.h
    main.c

nothing added to commit but untracked files present (use "git add" to track)
```

- 練習：作成したファイルをステージングエリアに追加しよう

git addでステージングエリアに追加する

- ファイル名が緑色になった！

```
2019-05-12 17:34:22 [kenta@Delphinium hello]
$ git add main.c
2019-05-12 17:34:28 [kenta@Delphinium hello]
$ git status
ブランチ master

No commits yet

コミット予定の変更点:
(use "git rm --cached <file>..." to unstage)

    new file:   main.c

追跡されていないファイル:
(use "git add <file>..." to include in what will be committed)

    Makefile
    hello.c
    hello.h
```

git addでステージングエリアに追加する (続)

- 実は"."ですべてのファイルを指定できる

```
2019-05-12 17:38:48 [kenta@Delphinium hello]
$ git add .
2019-05-12 17:38:49 [kenta@Delphinium hello]
$ git status
ブランチ master

No commits yet

コミット予定の変更点:
(use "git rm --cached <file>..." to unstage)

    new file:   Makefile
    new file:   hello.c
    new file:   hello.h
    new file:   main.c
```

git statusの表示はどういう意味？

ワーキングツリーに追加された
ファイルは
ステージングエリアに追加されたな

表示

```
2019-05-12 17:38:49 [kenta@Delphinium hello]
$ git status
ブランチ master

No commits yet

コミット予定の変更点:
(use "git rm --cached <file>..." to unstage)

    new file:   Makefile
    new file:   hello.c
    new file:   hello.h
    new file:   main.c
```

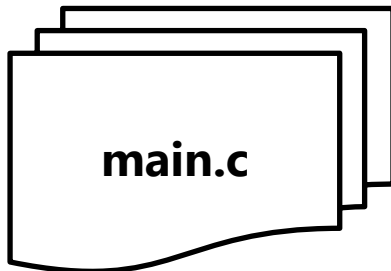
git statusの際に確認

ステージングエリアに追加された
ファイルは緑色で表示される

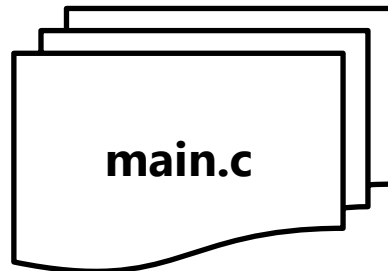
ワーキングツリー

ステージングエリア

リポジトリ



新規に作成されたファイル



addされたファイル

何もない

リポジトリにファイルを記録する

- commitコマンド：ステージングエリアのファイルをリポジトリに記録する
 - 使い方：git commit -m 'commit message'
 - コミットメッセージはなるべく「どんな変更を加えたか」が分かるように書こう
- 練習：リポジトリにファイルを記録しよう

```
2019-05-12 18:21:00 [kenta@Delphinium hello]
$ git commit -m 'first commit!'
[master (root-commit) d175fea] first commit!
4 files changed, 39 insertions(+)
create mode 100644 Makefile
create mode 100644 hello.c
create mode 100644 hello.h
create mode 100644 main.c
2019-05-12 18:21:09 [kenta@Delphinium hello]
$ git status
ブランチ master
nothing to commit, working tree clean
```

注意：commit時に-mを付けない場合

- エディタが起動してコミットメッセージが書けます
 - Ubuntu環境ではnano
- 私は(Vimmer|Emacser|VSCoder)だ!?!という貴方に
- `git config --global core.editor <editor>`して
落ち着いてください

```
.g/COMMIT_EDITMSG buffers
1
2 # Please enter the commit message for your changes. Lines starting
3 # with '#' will be ignored, and an empty message aborts the commit.
4 #
5 # ブランチ master
6 #
7 # 最初のコミット
8 #
9 # コミット予定の変更点:
10 #      new file:   Makefile
11 #      new file:   hello.c
12 #      new file:   hello.h
13 #      new file:   main.c
14 #
```

git statusの表示はどういう意味？

ワーキングツリーの内容は
リポジトリと一緒にだ

表示

```
2019-05-12 18:21:00 [kenta@Delphinium hello]
$ git commit -m 'first commit!'
[master (root-commit) d175fea] first commit!
4 files changed, 39 insertions(+)
create mode 100644 Makefile
create mode 100644 hello.c
create mode 100644 hello.h
create mode 100644 main.c
2019-05-12 18:21:09 [kenta@Delphinium hello]
$ git status
ブランチ master
nothing to commit, working tree clean
```

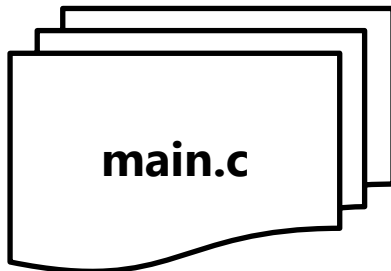
"nothing to commit"と表示される

git statusの際に確認

ワーキングツリー

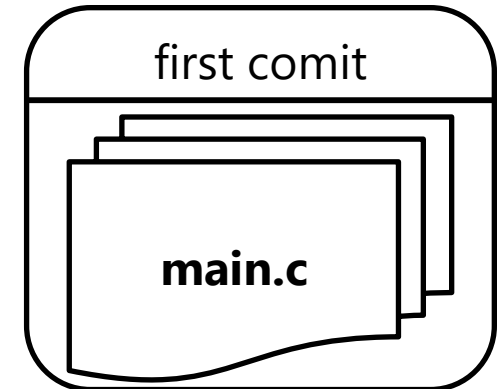
ステージングエリア

リポジトリ



commit済みのファイル

何もない



コミットログを確認する

- logコマンド：コミットログを表示する
- 練習：コミットログを見てみよう
 - configで設定した名前とメールアドレスが見られる
 - コミットメッセージが見られる

```
2020-04-04 00:36:43 [kenta@Delphinium source_code]
$ git log
commit e28451ca2d8a4b88b85042bcb6795b4622389475 (HEAD -> master)
Author: Kenta Arai <kenta@mymail>
Date:   Sat Apr 4 00:36:42 2020 +0900

First commit!
```

ざっと簡単なコミットログを見たい

- コミットが増えてくると見辛くなる
 - どんな変更があったか確認したいだけなのに, author情報とかいらないよね
- `git log --oneline`でコミットが一行ずつ表示される

```
commit b73fbc76c6fc446da90dd2cdac620155e37e5514 (HEAD -> master, tag: v8.1.1324, origin/master, origin/HEAD)
Author: Bram Moolenaar <Bram@vim.org>
Date: Sat May 11 21:50:07 2019 +0200

    patch 8.1.1324: stray comma in VMS makefile

    Problem: Stray comma in VMS makefile.
    Solution: Remove the comma. (Naruhiko Nishino, closes #4368)

commit 5d0183b706c618bf043380f7e995987cde9e7d56 (tag: v8.1.1323)
Author: Bram Moolenaar <Bram@vim.org>
Date: Sat May 11 21:38:58 2019 +0200

    patch 8.1.1323: 'mouse' option is reset when using GPM mouse

    Problem: 'mouse' option is reset when using GPM mouse.
    Solution: Add flag for GPM mouse.

commit 6e75e0a400d85cbcc27e2190ff448196bca025a8 (tag: v8.1.1322)
Author: Bram Moolenaar <Bram@vim.org>
Date: Sat May 11 21:24:26 2019 +0200

    patch 8.1.1322: Cygwin makefile is not nicely indented

    Problem: Cygwin makefile is not nicely indented.
    Solution: Addjust spaces in preprocessor directives. (Ken Takata)

commit a334772967de25764ed7b11d768e8b977818d0c6 (tag: v8.1.1321)
Author: Bram Moolenaar <Bram@vim.org>
Date: Sat May 11 21:14:24 2019 +0200

    patch 8.1.1321: no docs or tests for listener functions

    Problem: No docs or tests for listener functions.
    Solution: Add help and tests for listener_add() and listener_remove().
    Invoke the callbacks before redrawing.
```

```
b73fbc76c (HEAD -> master, tag: v8.1.1324, origin/master, origin/HEAD) patch 8.1.1324: stray comma i
n VMS makefile
5d0183b70 (tag: v8.1.1323) patch 8.1.1323: 'mouse' option is reset when using GPM mouse
6e75e0a40 (tag: v8.1.1322) patch 8.1.1322: Cygwin makefile is not nicely indented
a33477296 (tag: v8.1.1321) patch 8.1.1321: no docs or tests for listener functions
6d2399bdl (tag: v8.1.1320) patch 8.1.1320: it is not possible to track changes to a buffer
6ed881982 (tag: v8.1.1319) patch 8.1.1319: computing function length name in many places
ec28d1516 (tag: v8.1.1318) patch 8.1.1318: code for text changes is in a "misc" file
3f86ca0fa Add missing files from patch 8.1.1318
dc9f9e93f (tag: v8.1.1317) patch 8.1.1317: output from Travis can be improved
d6896731e (tag: v8.1.1316) patch 8.1.1316: duplicated localtime() call
afd78266c (tag: v8.1.1315) patch 8.1.1315: there is always a delay if a termrequest is never answer
d
c049b52b9 (tag: v8.1.1314) patch 8.1.1314: MSVC makefile is not nicely indented
63d2555c9 (tag: v8.1.1313) patch 8.1.1313: warnings for using localtime() and ctime()
4ca41534b (tag: v8.1.1312) patch 8.1.1312: Coverity warning for using uninitialized variable
23b513923 (tag: v8.1.1311) patch 8.1.1311: aborting an autocmd with an exception is not tested
42ae78c9f (tag: v8.1.1310) patch 8.1.1310: named function arguments are never optional
6b528fa06 (tag: v8.1.1309) patch 8.1.1309: test for Normal highlight fails on MS-Windows GUI
f90b6e03a (tag: v8.1.1308) patch 8.1.1308: the Normal highlight is not defined when compiled with GU
I
a6c27c47d Update runtime files
d4aa83af1 (tag: v8.1.1307) patch 8.1.1307: cannot reconnect to the X server after it restarted
eae1b91fe (tag: v8.1.1306) patch 8.1.1306: Borland support is outdated and doesn't work
691ddeefb (tag: v8.1.1305) patch 8.1.1305: there is no easy way to manipulate environment variables
68cbb14ba (tag: v8.1.1304) patch 8.1.1304: MS-Windows: compiler warning for unused value
be0a2597a (tag: v8.1.1303) patch 8.1.1303: not possible to hide a balloon
06bd82486 (tag: v8.1.1302) patch 8.1.1302: v:beval_text is not tested in Visual mode
0b75f7c97 (tag: v8.1.1301) patch 8.1.1301: when compiled with VIMDLL some messages are not shown
2f10658b0 (tag: v8.1.1300) patch 8.1.1300: in a terminal 'ballooneval' does not work right away
a5c6a0b6c (tag: v8.1.1299) patch 8.1.1299: "extends" from 'listchars' is used when 'list' is off
5416b7503 (tag: v8.1.1298) patch 8.1.1298: invalid argument test fails without X clipboard
240f7abab (tag: v8.1.1297) patch 8.1.1297: invalid argument test fails without GTK
27821260c (tag: v8.1.1296) patch 8.1.1296: crash when using invalid command line argument
98ffe4c6d (tag: v8.1.1295) patch 8.1.1295: when vimrun.exe does not exist external command may fail
93d77b2cb (tag: v8.1.1294) patch 8.1.1294: MS-Windows: Some fonts return wrong average char width
fda9784dc (tag: v8.1.1293) patch 8.1.1293: MSVC files are no longer useful
ba9ea91be (tag: v8.1.1292) patch 8.1.1292: invalid command line arguments not tested
1063f3d20 (tag: v8.1.1291) patch 8.1.1291: not easy to change directory and restore
```

大人気エディタVimのコミットログ

--onelineオプション付きのログ

.gitignoreで中間ファイルを無視する

- makeした後はstatusが以下のように表示される
- オブジェクトファイルと実行バイナリも表示される

```
2019-05-12 23:10:17 [kenta@Delphinium hello]
$ git status
ブランチ master
追跡されていないファイル:
  (use "git add <file>..." to include in what will be committed)

    hello
    hello.o
    main.o

nothing added to commit but untracked files present (use "git add" to track)
```

- commitすべきはソースコード
- 中間ファイルはコミットしない->容量を小さくできる
- .gitignore : commitしないファイルを指定する

.gitignoreを書く

- 練習
 - オブジェクトファイルと実行バイナリを無視するための.gitignoreを書く
 - .gitignoreをcommitする

```
.gitignore buffers
1 *.o
2 hello
```

```
2019-05-12 23:10:30 [kenta@Delphinium hello]
$ git status
ブランチ master
追跡されていないファイル:
  (use "git add <file>..." to include in what will be committed)

    .gitignore

nothing added to commit but untracked files present (use "git add" to track)
```

*.oとhelloが見えなくなるはず！

新しいバージョンのファイルを作る

- hello.(h|c)に新しく関数を追加し, main()で使用する
- ソースコードの例

hello.h

buffers

```
1 #ifndef __HELLO_H__
2 #define __HELLO_H__
3
4 void printHello(const char *name);
5 void printSeeYou(const char *name);
6
7 #endif // __HELLO_H__
```

hello.c buffers

```
1 #include "hello.h"
2
3 #include <stdio.h>
4
5 void printHello(const char *name) {
6     printf("Hello, %s!\n", name);
7 }
8
9 void printSeeYou(const char *name) {
10     printf("See you, %s!\n", name);
11 }
```

main.c buffers

```
1 #include "hello.h"
2
3 int
4 main(int argc, char* argv[]) {
5     printHello("Kenta");
6     printSeeYou("Kenta");
7
8     return 0;
9 }
```

新しいバージョンのファイルをcommitをする

- 練習
 - 変更をcommitしよう
 - .gitignoreの設定が反映されているかをaddする前に確認すること
 - 新しいコミットメッセージを確認しよう

```
2019-05-12 23:15:53 [kenta@Delphinium hello]
$ git log --oneline
af3a8cb (HEAD -> master) add printSeeYou()
8f71867 add .gitignore
d175fea first commit!
```

コミットログが3つ見えるはず

過去に戻って歴史を修正する

- commitした後に変更を修正したい場合がある
 - 今回のケース：hello.(h|c)の中にprintSeeYou()はねえだろう...
- resetコマンド：以前のコミットに戻る
 - 使い方 1 : `git reset --hard HEAD`(戻るコミット分だけ^)
 - 使い方 2 : `git reset --hard <commit number>`
- 練習
 - 一つ前のコミットに戻ってみよう
- ＊ 注意 ＊
 - resetのオプションにはsoftとmixもあるが、今回は触れない
 - 頻繁にresetするようなら、そもそも開発スタイルを見直そう
 - ブランチ（後述）を積極的に使う等

git reset : 使い方 1

- 一つ前に戻るので
git reset --hard HEAD^

```
2019-05-12 23:15:53 [kenta@Delphinium hello]
$ git log --oneline
af3a8cb (HEAD -> master) add printSeeYou()
8f71867 add .gitignore
d175fea first commit!
2019-05-12 23:15:56 [kenta@Delphinium hello]
$ git reset --hard HEAD^
HEAD is now at 8f71867 add .gitignore
2019-05-12 23:23:43 [kenta@Delphinium hello]
$ git log --oneline
8f71867 (HEAD -> master) add .gitignore
d175fea first commit!
```

git reset : 使い方 2

- 8f71867に戻るので
git reset --hard 8f71867

```
2019-05-12 23:25:54 [kenta@Delphinium hello]
$ git log --oneline
af3a8cb (HEAD -> master) add printSeeYou()
8f71867 add .gitignore
d175fea first commit!
2019-05-12 23:25:55 [kenta@Delphinium hello]
$ git reset --hard 8f71867
HEAD is now at 8f71867 add .gitignore
2019-05-12 23:26:08 [kenta@Delphinium hello]
$ git log --oneline
8f71867 (HEAD -> master) add .gitignore
d175fea first commit!
```


新しくファイルを追加する

- 新しいファイルをリポジトリに追加する場合でもaddしてcommitすればOK
- 練習
 - printSeeYou()を含むseeYou.(h|c)をコーディングしてリポジトリに反映しよう
 - main()でprintSeeYou()が使えるようにすること
 - 期待通りに実行できることを確認すること

注意事項

- MakefileにseeYou.cを追加すること

```
Makefile buffers
1 SRCS = \
2     hello.c \
3     seeYou.c \
4     main.c
5
6 OBJS = $(subst .c,.o,$(SRCS))
7 TARGET = hello
8
9 .SUFFIXES: .c .o
10
11 all : $(TARGET)
12
13 $(TARGET) : $(OBJS)
14     gcc -o $@ $(OBJS) $(LIBS)
15
16 .c.o :
17     gcc -c $(CFLAGS) -I. $< -o $@
18
19 clean :
20     rm -f *.o $(TARGET)
```

本章のまとめ

- Gitの設定
 - configで名前とメールアドレスを設定した
- リポジトリの作り方・使い方
 - init : リポジトリを作成
 - status : 各ファイルの状態を表示
 - add : ファイルをステージングエリアに追加
 - commit : ステージングエリアの内容をリポジトリに記録
 - log : コミットログを表示
 - reset : コミットを古いものに戻す

アウトライン

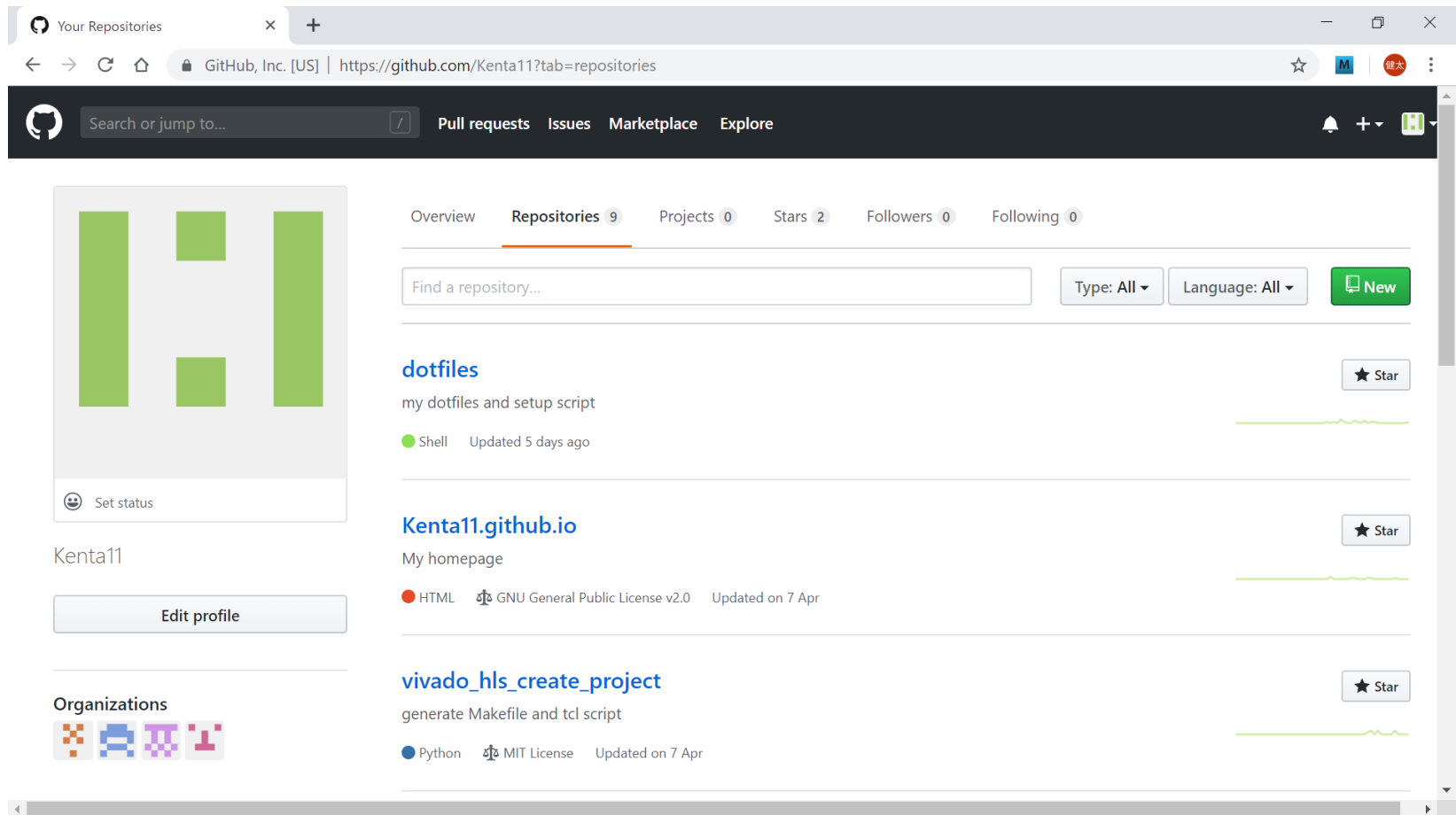
- Gitとは
- Gitの基礎
- リモートリポジトリを使う
- ブランチを使おう
- おまけ) Visual Studio CodeでGitしよう
- 更新履歴

リモートリポジトリ

- リモートリポジトリを使うケース
 - 複数人と共同で開発
 - 公開用のリポジトリにする
 - バックアップ
- 本章ですること
 - Githubにリモートリポジトリを作る
 - ローカルリポジトリの変更をリモートリポジトリに反映する
 - リモートリポジトリの変更をローカルリポジトリに反映する

Githubにリポジトリを作る

- 緑色の`New`ボタンを押す



リポジトリ名と説明を書く

Create a New Repository

GitHub, Inc. [US] | https://github.com/new

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?
[Import a repository.](#)

Owner

Repository name *

Kenta11


/ hello

Great repository names are short and memorable. Need inspiration? How about [cuddly-memory](#)?


Description (optional)

Tutorial repository for non-git user

☒

 **Public**
Anyone can see this repository. You choose who can commit.

☐


 **Private**
You choose who can see and commit to this repository.

☐

Initialize this repository with a README
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: None

Add a license: None



Create repository

Githubにリポジトリが出来ました！

The screenshot shows a web browser window displaying the GitHub repository page for 'Kenta11/hello'. The browser's address bar shows the URL 'https://github.com/Kenta11/hello'. The repository page header includes the repository name 'Kenta11 / hello' and statistics: 1 Unwatch, 0 Stars, and 0 Forks. Below the header, there are tabs for 'Code', 'Issues', 'Pull requests', 'Projects', 'Wiki', 'Insights', and 'Settings'. The main content area is titled 'Quick setup — if you've done this kind of thing before' and provides instructions for setting up the repository. It includes a 'Set up in Desktop' button, a link to 'HTTPS' (selected), and the repository URL 'https://github.com/Kenta11/hello.git'. Below this, it says 'Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.' The next section is titled '...or create a new repository on the command line' and shows a list of Git commands to initialize and push the repository. The final section is titled '...or push an existing repository from the command line' and shows the commands to add the remote and push the repository.

Kenta11/hello

GitHub, Inc. [US] | https://github.com/Kenta11/hello

Kenta11 / hello

Unwatch 1 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings

Quick setup — if you've done this kind of thing before

Set up in Desktop or HTTPS SSH https://github.com/Kenta11/hello.git

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

...or create a new repository on the command line

```
echo "# hello" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/Kenta11/hello.git
git push -u origin master
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/Kenta11/hello.git
git push -u origin master
```


ローカルリポジトリをアップロードする

- 指示通りにコマンドを入力

The screenshot shows the GitHub repository page for 'Kenta11/hello'. The browser address bar shows 'https://github.com/Kenta11/hello'. The repository page includes navigation tabs for 'Code', 'Issues', 'Pull requests', 'Projects', 'Wiki', 'Insights', and 'Settings'. The 'Quick setup' section offers options to 'Set up in Desktop' or 'HTTPS' with the URL 'https://github.com/Kenta11/hello.git'. Below this, the '...or create a new repository on the command line' section provides a list of Git commands to initialize and push a new repository. The final section, '...or push an existing repository from the command line', shows the commands to add a remote origin and push to the master branch. The commands in this section are highlighted with a red rectangle.

Quick setup — if you've done this kind of thing before

or

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

...or create a new repository on the command line

```
echo "# hello" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/Kenta11/hello.git
git push -u origin master
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/Kenta11/hello.git
git push -u origin master
```

アップロードの様子

- Githubのユーザ名とパスワードを求められるので入力すること

```
2019-05-12 23:37:29 [kenta@Delphinium hello]
$ git remote add origin https://github.com/Kenta11/hello.git
2019-05-12 23:57:36 [kenta@Delphinium hello]
$ git push -u origin master
Username for 'https://github.com': Kenta11
Password for 'https://Kenta11@github.com':
Counting objects: 15, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (14/14), done.
Writing objects: 100% (15/15), 1.58 KiB | 55.00 KiB/s, done.
Total 15 (delta 3), reused 0 (delta 0)
remote: Resolving deltas: 100% (3/3), done.
To https://github.com/Kenta11/hello.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
```

プロキシの設定

- プロキシによってアップロードが阻まれることがある
- その場合はconfigを使ってプロキシの設定をする

```
2020-04-04 00:24:11 [kenta@Delphinium ~]  
$ git config --global http.proxy http://myproxy.com:8888  
2020-04-04 00:24:33 [kenta@Delphinium ~]  
$ git config --global https.proxy http://myproxy.com:8080
```

Githubにリポジトリを見に行こう！

- 表示されない場合はブラウザをリロードすること

The screenshot shows a web browser window displaying the GitHub repository page for 'Kenta11/hello'. The browser's address bar shows the URL 'https://github.com/Kenta11/hello'. The repository page includes a header with the repository name, navigation tabs (Code, Issues, Pull requests, Projects, Wiki, Insights, Settings), and statistics (1 star, 0 forks). Below the header, there's a section for 'Tutorial repository for non-git user' with an 'Edit' button. A progress bar shows 3 commits, 1 branch, 0 releases, and 0 contributors. A 'Branch: master' dropdown and a 'New pull request' button are visible. A 'Clone or download' button is also present. The commit history table shows the latest commit by Kenta Arai, adding 'printSeeYou()' 25 minutes ago. The table lists files: .gitignore, Makefile, hello.c, hello.h, main.c, seeYou.c, and seeYou.h, each with its commit message and time. At the bottom, there's a prompt to 'Add a README'.

Kenta11 / hello

Unwatch 1 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings

Tutorial repository for non-git user Edit

Manage topics

3 commits 1 branch 0 releases 0 contributors

Branch: master New pull request Create new file Upload files Find File Clone or download

Kenta Arai add printSeeYou()	Latest commit 8f199ae 25 minutes ago
.gitignore	add .gitignore an hour ago
Makefile	add printSeeYou() 25 minutes ago
hello.c	first commit! 6 hours ago
hello.h	first commit! 6 hours ago
main.c	add printSeeYou() 25 minutes ago
seeYou.c	add printSeeYou() 25 minutes ago
seeYou.h	add printSeeYou() 25 minutes ago

Help people interested in this repository understand your project by adding a README. Add a README

すでにあるリポジトリをダウンロードする

- clone : リモートリポジトリをダウンロードする
- 使い方
 - `git clone <url>`
- 練習
 - リポジトリを削除して, Githubからダウンロードしてみよう

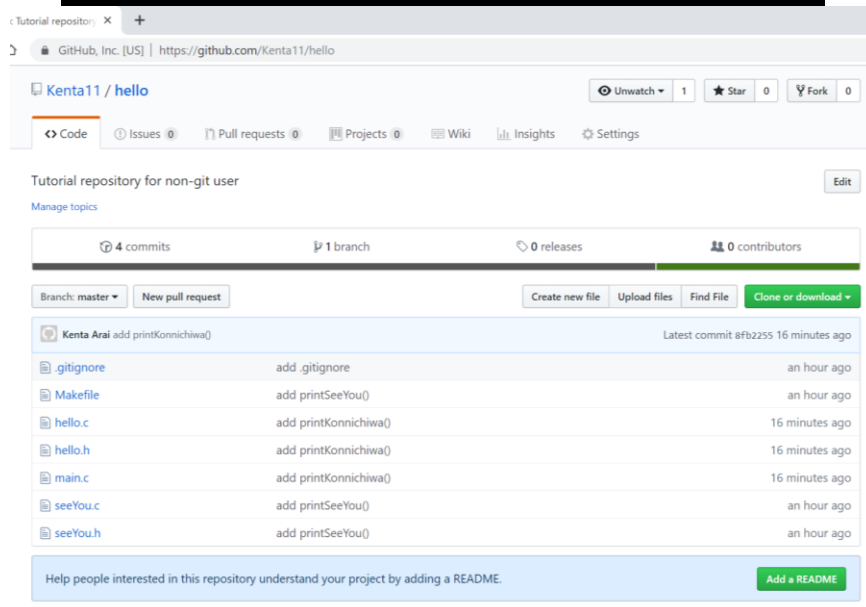
```
2019-05-13 00:05:19 [kenta@Delphinium git]
$ rm -rf hello
2019-05-13 00:05:22 [kenta@Delphinium git]
$ ls
Fast-RTPS  OpenCL-FPGA-examples  vim  vivado_hls_create_project
2019-05-13 00:05:26 [kenta@Delphinium git]
$ git clone https://github.com/Kenta11/hello
Cloning into 'hello'...
remote: Enumerating objects: 15, done.
remote: Counting objects: 100% (15/15), done.
remote: Compressing objects: 100% (11/11), done.
remote: Total 15 (delta 3), reused 15 (delta 3), pack-reused 0
Unpacking objects: 100% (15/15), done.
```

ローカルの変更をリモートに反映する(1/2)

- push : リポジトリをリモートにアップロードする
- 使い方
 - git push
- 練習
 - hello.(h|c)にprintKonnichiwa()を追加してcommitする
 - 新しいcommitをリモートにアップロードする

ローカルの変更をリモートに反映する(2/2)

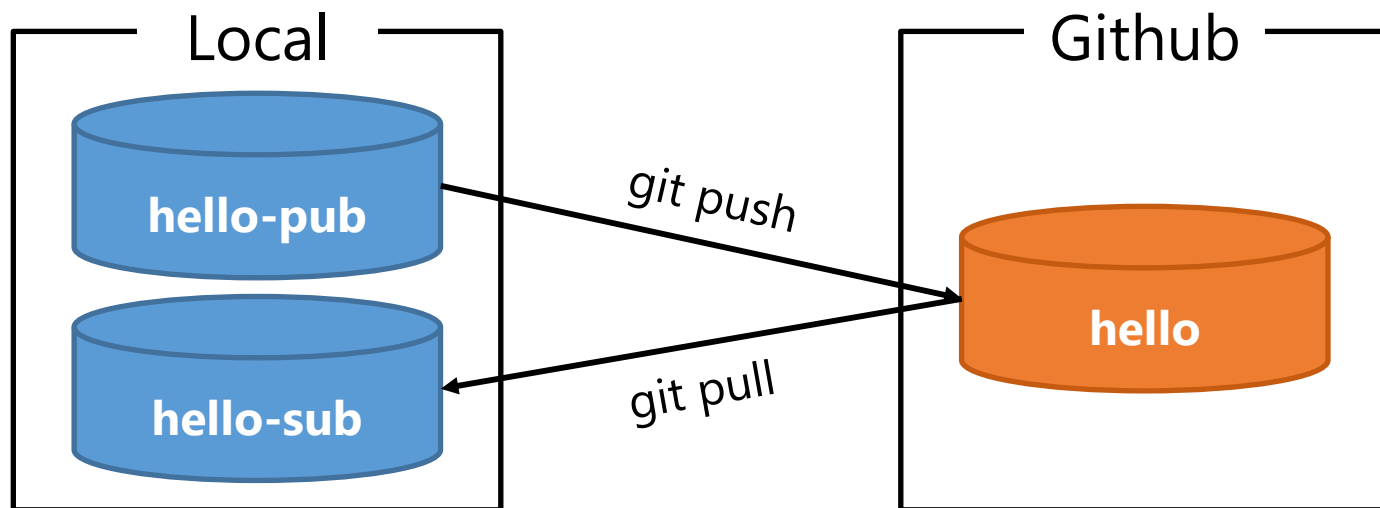
```
2019-05-13 00:17:25 [kenta@Delphinium hello]
$ git add .
2019-05-13 00:17:28 [kenta@Delphinium hello]
$ git commit -m 'add printKonnichiwa()'
[master 8fb2255] add printKonnichiwa()
3 files changed, 6 insertions(+)
2019-05-13 00:17:41 [kenta@Delphinium hello]
$ git push
Username for 'https://github.com': Kenta11
Password for 'https://Kenta11@github.com':
Counting objects: 5, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 702 bytes | 87.00 KiB/s, done.
Total 5 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/Kenta11/hello
8f199ae..8fb2255 master -> master
```



- add & commitしてコミットを追加
- pushでコミットをアップロード
- コミット数が増えている

リモートの変更をローカルに反映する(1/3)

- pull : ローカルリポジトリをリモートのものに更新
- 使い方
 - git pull
- 練習の準備
 - 更新用リポジトリを準備する
 - 複数人での開発を模擬



リモートの変更をローカルに反映する(2/3)

- hello-pub, hello-subを作る
 - 中身はhello

```
2019-05-13 01:14:25 [kenta@Delphinium hello]
$ cd ..
2019-05-13 01:14:26 [kenta@Delphinium git]
$ mv hello hello-pub
2019-05-13 01:14:31 [kenta@Delphinium git]
$ git clone https://github.com/Kenta11/hello
Cloning into 'hello'...
remote: Enumerating objects: 20, done.
remote: Counting objects: 100% (20/20), done.
remote: Compressing objects: 100% (15/15), done.
remote: Total 20 (delta 6), reused 18 (delta 4), pack-reused 0
Unpacking objects: 100% (20/20), done.
2019-05-13 01:14:44 [kenta@Delphinium git]
$ mv hello hello-sub
2019-05-13 01:14:50 [kenta@Delphinium git]
$ ls
Fast-RTPS  OpenCL-FPGA-examples  hello-pub  hello-sub  vim  vivado_hls_create_projec
```

リモートの変更をローカルに反映する(3/3)

• 練習

- hello-pubでhello.(h|c)にprintNeHao()を追加しcommitする
- 新しいコミットをpushする
- hello-subでpullする

```
2019-05-13 01:42:21 [kenta@Delphinium hello-pub]
$ git add .
2019-05-13 01:42:23 [kenta@Delphinium hello-pub]
$ git commit -m 'add printNeHao()'
[master eccb3a1] add printNeHao()
3 files changed, 7 insertions(+)
2019-05-13 01:42:36 [kenta@Delphinium hello-pub]
$ git push
Username for 'https://github.com': Kenta11
Password for 'https://Kenta11@github.com':
Counting objects: 5, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 608 bytes | 121.00 KiB/s, done.
Total 5 (delta 3), reused 0 (delta 0)
remote: Resolving deltas: 100% (3/3), completed with 3 local objects.
To https://github.com/Kenta11/hello
8fb2255..ecb3a1 master -> master
```

```
2019-05-13 01:42:49 [kenta@Delphinium hello-pub]
$ cd ../hello-sub
2019-05-13 01:42:56 [kenta@Delphinium hello-sub]
$ git pull
remote: Enumerating objects: 9, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 5 (delta 3), reused 5 (delta 3), pack-reused 0
Unpacking objects: 100% (5/5), done.
From https://github.com/Kenta11/hello
8fb2255..ecb3a1 master -> origin/master
Updating 8fb2255..ecb3a1
Fast-forward
 hello.c | 4 ++++
 hello.h | 1 +
 main.c | 2 ++
3 files changed, 7 insertions(+)
```

本章のまとめ

- Githubにリモートリポジトリを作った
- リモートリポジトリをGithubにアップロードした
 - remote : リモートリポジトリの設定
- リモートリポジトリをダウンロードした
 - clone : リモートリポジトリをクローンする
- リモートリポジトリとローカルリポジトリの同期をした
 - push : リモートにリポジトリをアップロード
 - pull : ローカルにリモートの変更を反映

アウトライン

- Gitとは
- Gitの基礎
- リモートリポジトリを使う
- ブランチを使おう
- おまけ) Visual Studio CodeでGitしよう
- 更新履歴

branchを使って開発を分担する

- 今までのバージョン管理
 - ファイルを更新する度にコミットを作った
 - 問題：異なる種類の実装が同居できない
 - 問題が起きる例) 複数人で開発する, 試験的な実装を作る
- branch : コミットの流れを分岐する概念
- 本章で学ぶこと
 - ブランチを作成して開発を分ける
 - ブランチを統合する
 - ブランチをローカルとリモートでそれぞれ反映する

branchを見る

- branchコマンド：リポジトリのブランチを操作する
- 使い方
 - git branch
- 練習
 - helloリポジトリのブランチを表示しよう
 - masterが表示されるはず
 - masterは最初にできるブランチ

```
2019-05-13 12:17:11 [kenta@Delphinium hello-pub]  
$ git branch  
* master
```

リポジトリをbranchで分ける

- checkoutコマンド：ブランチの操作や移動
- 使い方：ブランチを作成して移動する
 - checkout --b <branch name>
- 練習
 - ブランチを作ってみよう
 - helloリポジトリにおけるブランチ作成の方針
 - masterブランチ：完成品を公開する
 - developブランチ：開発中の機能を随時置く

```
2019-05-13 12:23:25 [kenta@Delphinium hello-pub]
$ git checkout -b develop
Switched to a new branch 'develop'
2019-05-13 12:23:27 [kenta@Delphinium hello-pub]
$ git branch
* develop
master
```

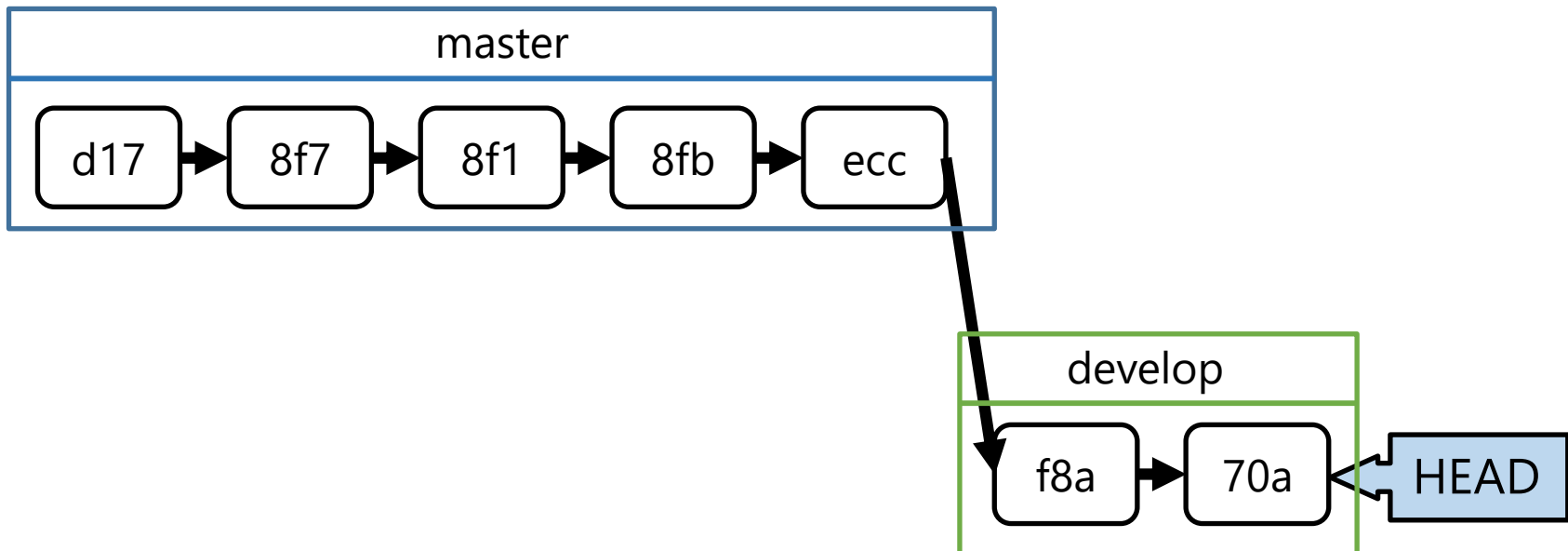
developブランチで開発する

- 練習
 - seeYou.(h|c)に以下の機能を追加し、それぞれコミットしよう
 - 機能 1 : printSayounara()
 - 機能 2 : printTsaichen()

```
2019-05-13 12:35:34 [kenta@Delphinium hello-pub]
$ git log --oneline
70a2b8a (HEAD -> develop) add printTsaichen()
f8af46d add printSayounara()
eccb3a1 (origin/master, origin/HEAD, master) add printNeHao()
8fb2255 add printKonnichiwa()
8f199ae add printSeeYou()
8f71867 add .gitignore
d175fea first commit!
```


ブランチとコミットの様子

```
2019-05-13 12:35:34 [kenta@Delphinium hello-pub]
$ git log --oneline
70a2b8a (HEAD -> develop) add printTsaichen()
f8af46d add printSayounara()
eccb3a1 (origin/master, origin/HEAD, master) add printNeHao()
8fb2255 add printKonnichiwa()
8f199ae add printSeeYou()
8f71867 add .gitignore
d175fea first commit!
```



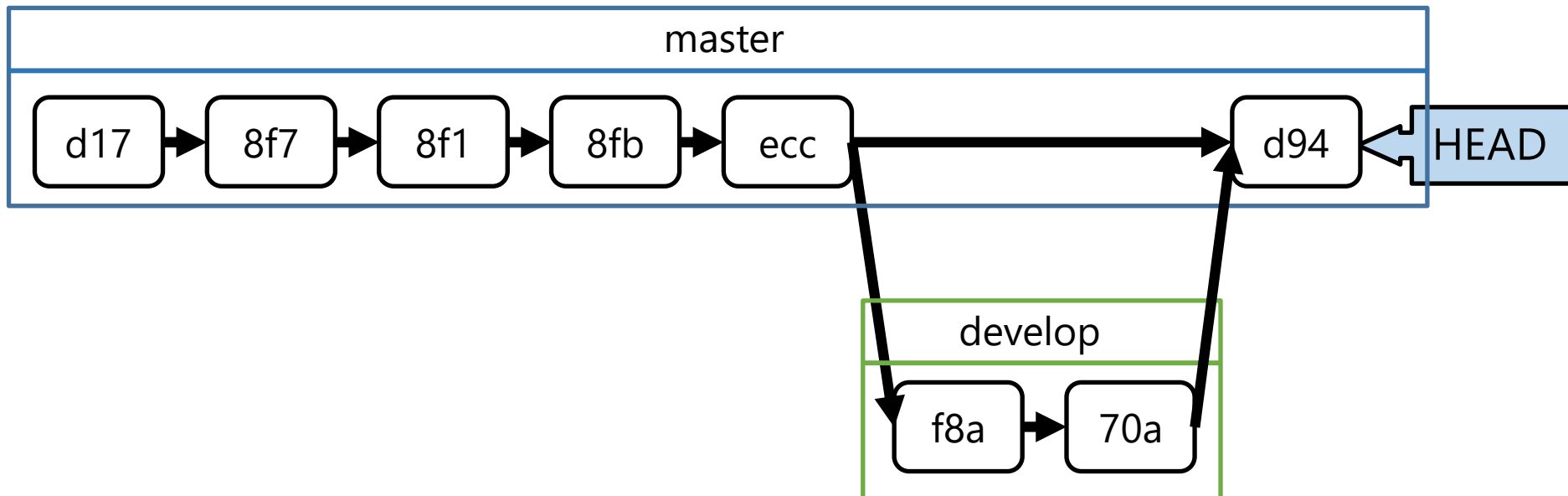
masterにdevelopの変更を統合する

- mergeコマンド：ブランチを統合する
- 使い方
 - git merge <branch name>
- 練習
 - masterにdevelopの変更を統合しよう

```
2019-05-13 18:38:51 [kenta@Delphinium hello-pub]
$ git log --all --oneline
70a2b8a (develop) add printTsaichen()
f8af46d add printSayounara()
eccb3a1 (HEAD -> master, origin/master, origin/HEAD) add printNeHao()
8fb2255 add printKonnichiwa()
8f199ae add printSeeYou()
8f71867 add .gitignore
d175fea first commit!
2019-05-13 18:38:54 [kenta@Delphinium hello-pub]
$ git merge --no-ff develop
Merge made by the 'recursive' strategy.
 main.c | 2 ++
 seeYou.c | 8 +++++++
 seeYou.h | 2 ++
 3 files changed, 12 insertions(+)
```

ブランチとコミットの様子

```
2019-05-13 18:44:11 [kenta@Delphinium hello-pub]
$ git log --graph --all --format="%x09%an%x09%h %d %s" (git)[*]
*      Kenta Arai      d94cde1  (HEAD -> master) add printSayounara() and printTsaichen()
| \
| *      Kenta Arai      70a2b8a  (develop) add printTsaichen()
| *      Kenta Arai      f8af46d  add printSayounara()
| /
*      Kenta Arai      eccb3a1  (origin/master, origin/HEAD) add printNeHao()
*      Kenta Arai      8fb2255  add printKonnichiwa()
*      Kenta Arai      8f199ae  add printSeeYou()
*      Kenta Arai      8f71867  add .gitignore
*      Kenta Arai      d175fea  first commit!
```

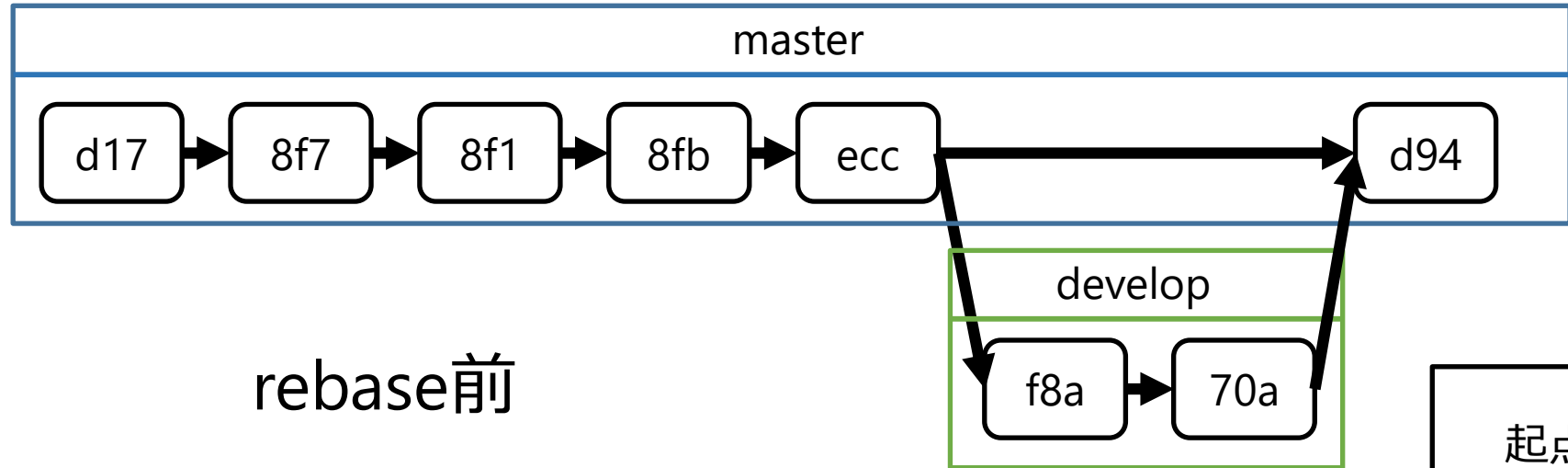


変更元のブランチが変更されていたら？

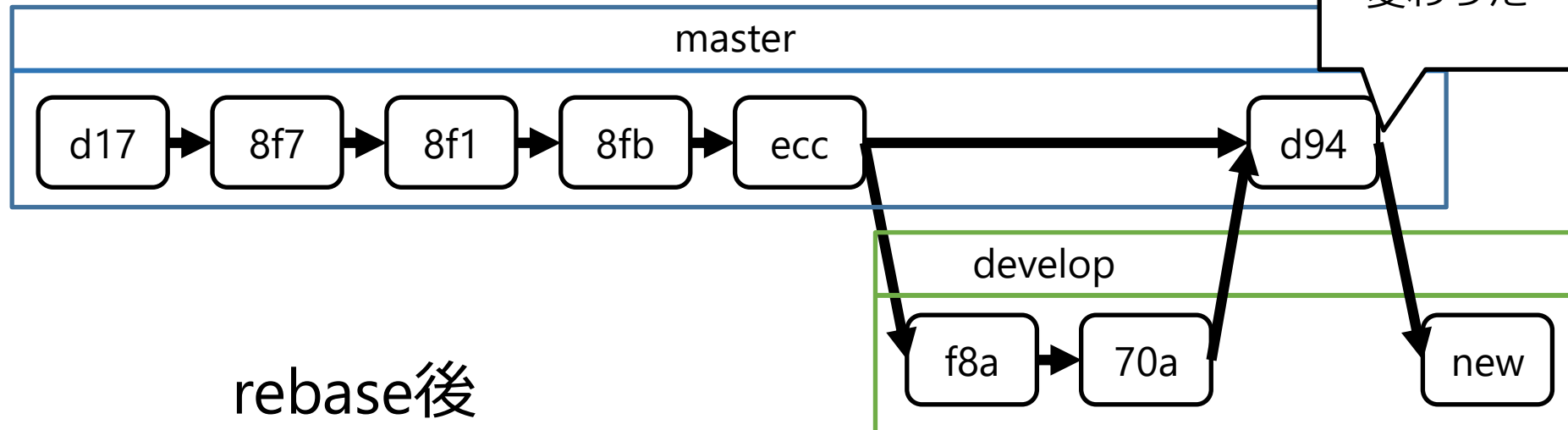
- rebaseコマンド：ブランチのベースを変更する
- 使い方
 - `git rebase <branch name>`
- 什么时候に使う？
 - 元のブランチにコミットが追加された場合
 - 自身のブランチが元のブランチにマージされた場合
- 練習
 - `git log --graph --oneline`でリベース前の状態を確認しよう
 - developにmasterをrebaseしよう
 - developブランチで``git rebase master``
 - `git log --graph --oneline`でリベース後の状態を確認しよう

git rebaseされたブランチ

rebase前



rebase後



変更は時にコンフリクトする

- 複数のブランチからマージをする場合は変更のコンフリクトに注意しなければならない
- 練習の準備
 - コンフリクトするブランチを作成しよう

```
2019-05-13 19:02:29 [kenta@Delphinium hello-pub]
$ git branch
  develop
* master
2019-05-13 19:02:30 [kenta@Delphinium hello-pub]
$ git checkout -b develop2
Switched to a new branch 'develop2'
2019-05-13 19:02:42 [kenta@Delphinium hello-pub]
$ git branch
  develop
* develop2
  master
```

コンフリクトさせる

- 練習
 - 各ブランチで同じ名前の関数を追加する
 - develop
 - printGreeting()を実装し, "Hello, %s!"と表示させる
 - develop2
 - printGreeting()を実装し, "Good morning, %s!"と表示させる
 - それぞれのコミットをmasterにマージする

```
2019-05-14 15:41:19 [kenta@Delphinium hello-pub]
$ git log --graph --oneline --all
* 240a60d (develop) add printGreeting()
| * 0622fd5 (develop2) add printGreeting function
|/
* d94cde1 (HEAD -> master) add printSayounara() and printTsaichen()
| \
| * 70a2b8a add printTsaichen()
| * f8af46d add printSayounara()
|/
* eccb3a1 (origin/master, origin/HEAD) add printNeHao()
* 8fb2255 add printKonnichiwa()
* 8f199ae add printSeeYou()
* 8f71867 add .gitignore
* d175fea first commit!
```

図：merge前のブランチ

developブランチのマージ

```
2019-05-14 15:43:46 [kenta@Delphinium hello-pub]
```

```
$ git log --graph --oneline --all
```

```
(git)[* master]
```

```
* 240a60d (develop) add printGreeting()
```

```
| * 0622fd5 (develop2) add printGreeting function
```

```
|/
```

```
* d94cde1 (HEAD -> master) add printSayounara() and printTsaichen()
```

```
| \
```

```
| * 70a2b8a add printTsaichen()
```

```
| * f8af46d add printSayounara()
```

```
|/
```

```
* eccb3a1 (origin/master, origin/HEAD) add printNeHao()
```

```
* 8fb2255 add printKonnichiwa()
```

```
* 8f199ae add printSeeYou()
```

```
* 8f71867 add .gitignore
```

```
* d175fea first commit!
```

```
2019-05-14 15:43:47 [kenta@Delphinium hello-pub]
```

```
$ git merge --no-ff develop
```

```
(git)[* master]
```

```
Merge made by the 'recursive' strategy.
```

```
hello.c | 4 ++++
```

```
hello.h | 1 +
```

```
main.c | 1 +
```

```
3 files changed, 6 insertions(+)
```


developブランチのマージ後

```
2019-05-14 15:44:12 [kenta@Delphinium hello-pub]
```

```
$ git log --graph --oneline --all
```

```
(git)[* master]
```

```
* 046f300 (HEAD -> master) merge from develop
```

```
| \
```

```
| * 240a60d (develop) add printGreeting()
```

```
| /
```

```
| * 0622fd5 (develop2) add printGreeting function
```

```
| /
```

```
* d94cde1 add printSayounara() and printTsaichen()
```

```
| \
```

```
| * 70a2b8a add printTsaichen()
```

```
| * f8af46d add printSayounara()
```

```
| /
```

```
* eccb3a1 (origin/master, origin/HEAD) add printNeHao()
```

```
* 8fb2255 add printKonnichiwa()
```

```
* 8f199ae add printSeeYou()
```

```
* 8f71867 add .gitignore
```

```
* d175fea first commit!
```

develop2ブランチのマージ

```
2019-05-14 15:46:57 [kenta@Delphinium hello-pub]
$ git merge --no-ff develop2
Auto-merging hello.c
CONFLICT (content): Merge conflict in hello.c
Automatic merge failed; fix conflicts and then commit the result.
```

- CONFLICTが発生したことが表示される
 - hello.cがおかしいようだ
 - このときマージは完了しない.

コンフリクトが起きたファイルを修正する

- developとdevelop2の内容がそれぞれ反映されている

```
17 void printGreeting(const char *name) {  
18 <<<<<<< HEAD  
19     printf("Hello, %s!\n", name);  
20 =====  
21     printf("Good morning, %s!\n", name);  
22 >>>>>>> develop2  
23 }
```

- マージしたい内容を残そう

```
17 void printGreeting(const char *name) {  
18     printf("Good morning, %s!\n", name);  
19 }
```

差分を見る

```
2019-05-14 15:53:27 [kenta@Delphinium hello-pub]
$ git diff master..develop2
diff --git a/hello.c b/hello.c
index 672b776..a113855 100644
--- a/hello.c
+++ b/hello.c
@@ -15,5 +15,5 @@ void printNeHao(const char *name) {
 }

void printGreeting(const char *name) {
-   printf("Hello, %s!\n", name);
+   printf("Good morning, %s!\n", name);
}

2019-05-14 15:54:41 [kenta@Delphinium hello-pub]
$ git diff 046f300 0622fd5
diff --git a/hello.c b/hello.c
index 672b776..a113855 100644
--- a/hello.c
+++ b/hello.c
@@ -15,5 +15,5 @@ void printNeHao(const char *name) {
 }

void printGreeting(const char *name) {
-   printf("Hello, %s!\n", name);
+   printf("Good morning, %s!\n", name);
}
```

- diffコマンド：CONFLICTした箇所を確認する
- 使い方
 - git diff <branch> or <commit>
 - 比較対象のブランチ名またはコミット名を2つ用意する

修正をコミットする(1/2)

```
2019-05-14 15:55:14 [kenta@Delphinium hello-pub]
$ git status
ブランチ master
このブランチは 'origin/master' よりも5コミット進んでいます。
(use "git push" to publish your local commits)

You have unmerged paths.
(fix conflicts and run "git commit")
(use "git merge --abort" to abort the merge)

Unmerged paths:
(use "git add <file>..." to mark resolution)

        both modified:   hello.c

no changes added to commit (use "git add" and/or "git commit -a")
2019-05-14 15:59:02 [kenta@Delphinium hello-pub]
$ git add hello.c
2019-05-14 15:59:38 [kenta@Delphinium hello-pub]
$ git status
ブランチ master
このブランチは 'origin/master' よりも5コミット進んでいます。
(use "git push" to publish your local commits)

All conflicts fixed but you are still merging.
(use "git commit" to conclude merge)

コミット予定の変更点:

        modified:   hello.c
```

- CONFLICTしたファイルはstatusでも確認できる
- 修正したファイルをadd

修正をコミットする(2/2)

```
2019-05-14 15:59:42 [kenta@Delphinium hello-pub]
$ git commit -m 'modify printGreeting()'
[master a00d420] modify printGreeting()
2019-05-14 16:01:49 [kenta@Delphinium hello-pub]
$ git log --graph --all --oneline
*   a00d420 (HEAD -> master) modify printGreeting()
| \
| * 0622fd5 (develop2) add printGreeting function
* |   046f300 merge from develop
| \ \
| | /
| / |
| * 240a60d (develop) add printGreeting()
| /
| * d94cde1 add printSayounara() and printTsaichen()
| \
| * 70a2b8a add printTsaichen()
| * f8af46d add printSayounara()
| /
* eccb3a1 (origin/master, origin/HEAD) add printNeHao()
* 8fb2255 add printKonnichiwa()
* 8f199ae add printSeeYou()
* 8f71867 add .gitignore
* d175fea first commit!
```

- logを表示してみよう
- 引き続きdevelopで開発するときは, rebaseを忘れずに

リモートリポジトリにブランチをpushする

- push origin : リモートにブランチをアップロード
- 使い方
 - `git push origin <branch>`
- 練習
 - Githubにdevelopをアップロードしよう
 - Githubでdevelopブランチを確認しよう
 - 単に`git push`した場合との違いを比較しよう

本章のまとめ

- ブランチの作り方と使い方を学んだ
 - branch : ブランチを表示
 - checkout : ブランチを操作する
 - rebase : ベースを変更する
- ブランチのマージの仕方を学んだ
 - merge : ブランチをマージ
 - diff : 差分を調べる
- 新しいブランチをリモートに反映する方法を学んだ
 - push origin : リモートにブランチをコミットする

アウトライン

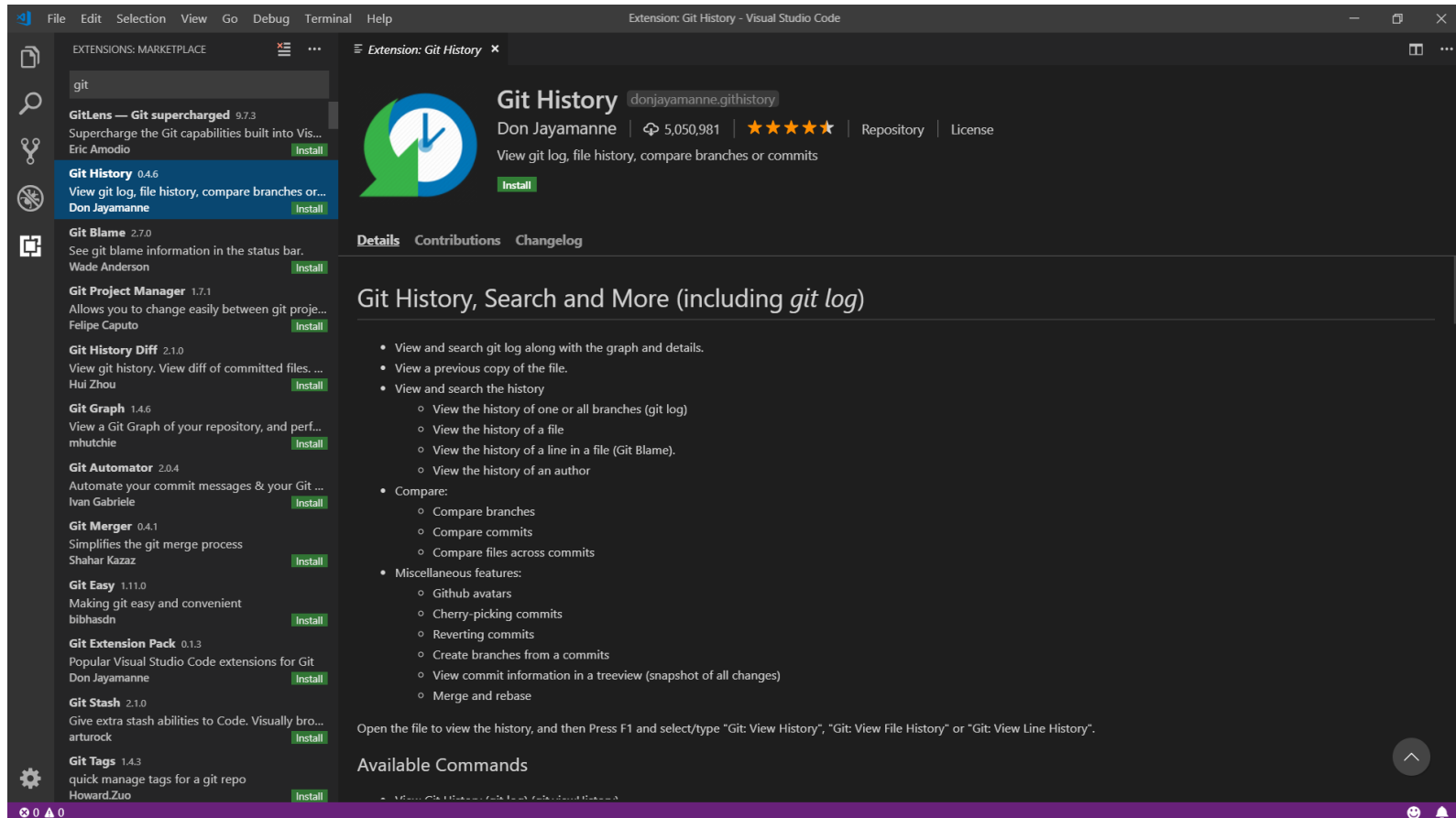
- Gitとは
- Gitの基礎
- リモートリポジトリを使う
- ブランチを使おう
- おまけ) Visual Studio CodeでGitしよう
- 更新履歴

おまけ) VSCodeでGitしよう

- VSCodeのGitプラグインを使うとグラフィカルにGitを操作できる
- 本章で学ぶこと
 - VSCodeにプラグインをインストールする方法
 - Gitプラグインを使ったリポジトリの操作

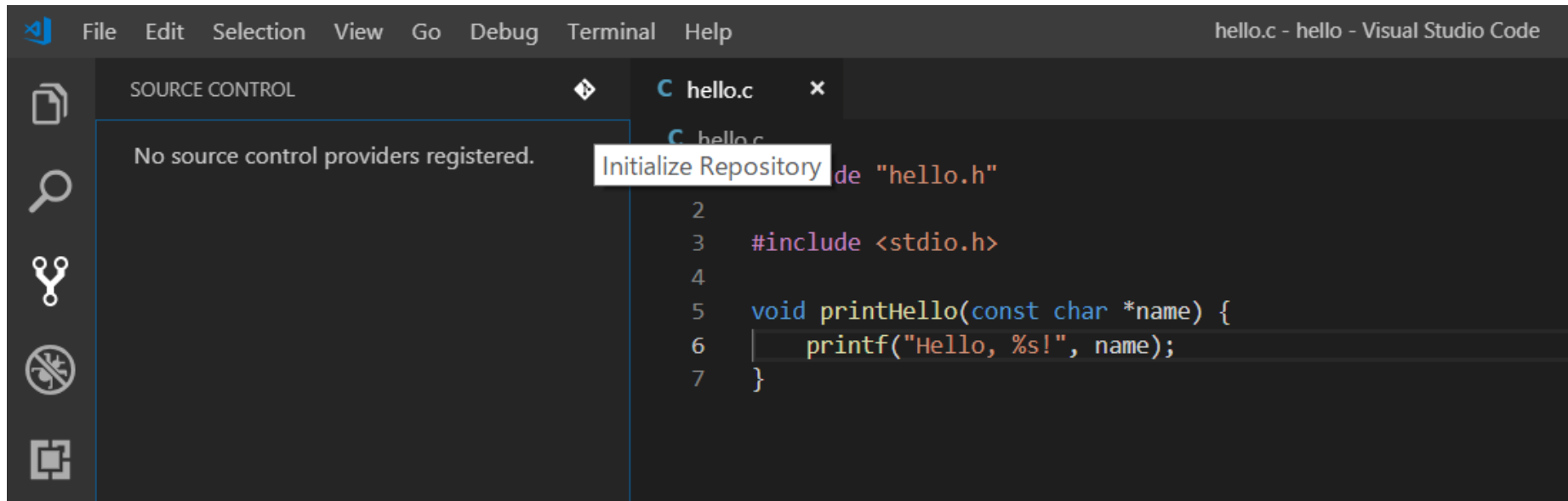
Git Historyをインストールする

- Extensionsをクリック
- `Search Extensions in Marketplace`にgitと入力



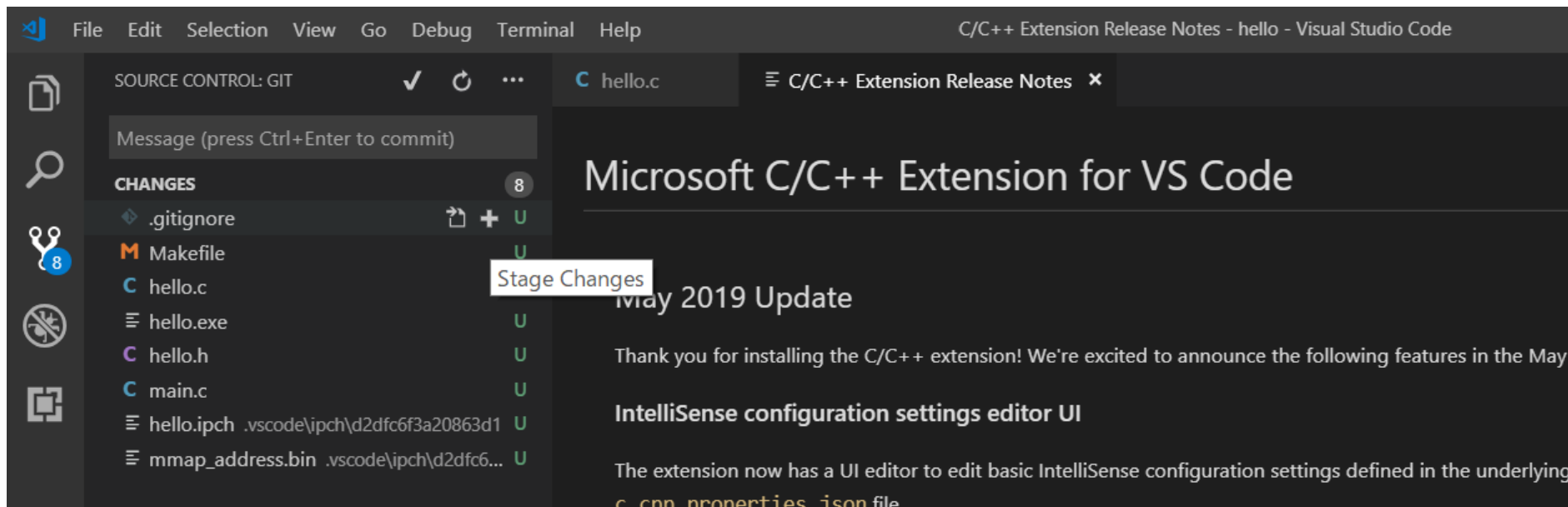
新しくリポジトリを作ってみよう(git init)

- helloディレクトリにCプログラムとMakefileをコピー
 - Gitの基礎と同じ手順で
- `Source Control`をクリック
- `Initialize Repository`でリポジトリを作成
 - リポジトリを作成するディレクトリを指定する



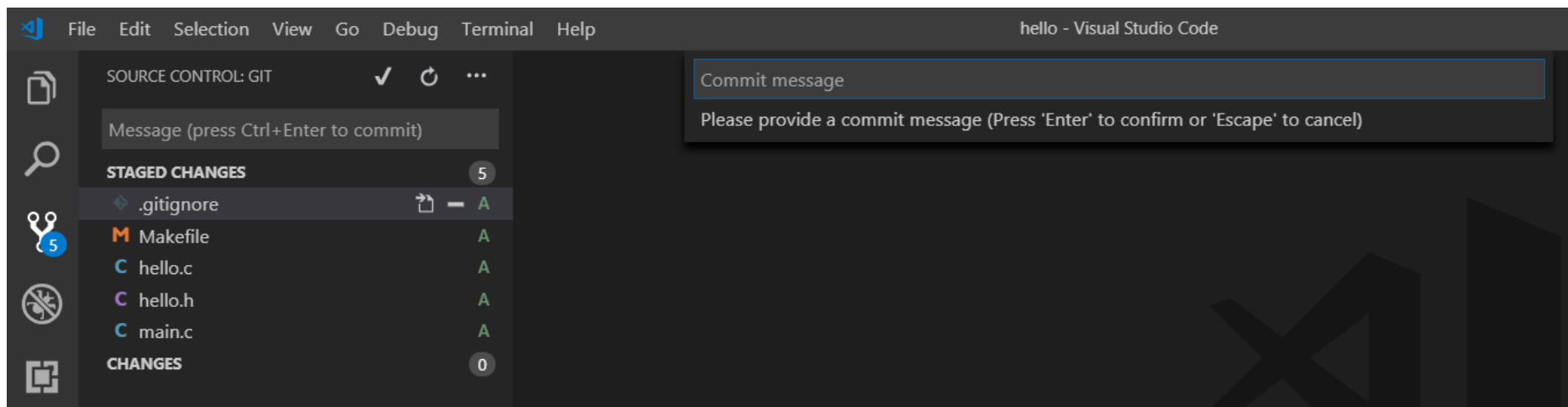
ステージングエリアにファイルを追加(git add)

- ``+``でファイルをステージングエリアに追加
 - .vscode/*も見えている場合は.gitignoreにルールを追加



コミットする(git commit)

- チェックマークをクリック
- `Commit message`にコミットメッセージを入力



More Information

- Qiitaの記事が詳しい
- <https://qiita.com/y-tsutsu/items/2ba96b16b220fb5913be>

The screenshot shows a web browser displaying a Qiita article. The browser's address bar shows the URL <https://qiita.com/y-tsutsu/items/2ba96b16b220fb5913be>. The Qiita header is green with the logo, navigation links (ホーム, コミュニティ), a search bar, and buttons for 'ストック一覧', '投稿する', and a user profile icon. On the left sidebar, there are icons for likes (688), bookmarks, comments (1), and social media links. The article itself is by user '@y-tsutsu' and was updated on '2019年04月24日'. The title is 'VSCodeでのGitの基本操作まとめ'. Below the title are tags: 'Git', 'VSCode', 'VisualStudioCode', and '新人プログラマ応援'. The article content begins with the heading 'はじめに' followed by a paragraph: '一年前に新人研修でGitを担当してTigの記事を書いたのですが、今年も同じくGitの研修を担当することになりました。新人さんたちにとってはターミナル環境はとっつきにくい人も多いようで、短い研修期間では操作自体に苦戦してしまい、Gitそのものを理解するということに力を割けない人も少なくあ'. On the right, a vertical list of links is visible: 'はじめに', 'インストール', 'Gitの初期設定', 'VSCodeの起動', 'リポジトリ作成', and 'コミットしてみる'.

VSCodeでのGitの基本操作まとめ - ×

← → ↺ 🏠 <https://qiita.com/y-tsutsu/items/2ba96b16b220fb5913be> ☆ M 健太

Qiita ホーム コミュニティ 🔍 キーワードを入力

📁 スtock一覧 🖋️ 投稿する 0 🧑

688 👍

📁 1 💬 🐦 f

@y-tsutsu 2019年04月24日に更新

VSCodeでのGitの基本操作まとめ

Git VSCode VisualStudioCode 新人プログラマ応援

はじめに

一年前に新人研修でGitを担当してTigの記事を書いたのですが、今年も同じくGitの研修を担当することになりました。新人さんたちにとってはターミナル環境はとっつきにくい人も多いようで、短い研修期間では操作自体に苦戦してしまい、Gitそのものを理解するということに力を割けない人も少なくあ

はじめに
インストール
Gitの初期設定
VSCodeの起動
リポジトリ作成
コミットしてみる

更新履歴

- Ver1.0 : 初リリース