WEB+DBシステム（応用編）

第2回(2016年9月29日)

Gitを使う

今日の目標

Gitによって、バックアップを作成する。
バージョン管理の考え方を学ぶ。

Gitの使い方を学ぶ。

(1) 誤った修正を記録し、それを後から取り除く。
(2) ソースコードを、バックアップした状態に戻す。
(3) 過去に行った操作を、自動で反復して復元する。

Gitの参考サイト

私の教材(なんか)よりも、はるかにパワフルなサイトが
　無数にあります。

例えば:

http://sourceforge.jp/magazine/09/03/16/0831212

（Gitを使いこなすための20のコマンド）

http://morizyun.github.io/blog/how-to-git-review-book/

（チーム開発に必要なgitコマンドを神速で習得しよう!）
　↑このタイトルは「釣り」だそうですが・・・

この人のスライドに乗ります！

http://www.slideshare.net/zephiransas/git-14810093

(一人でもはじめるGitでバージョン管理)

Gitが特に役に立つ場面

Githubなどにあるオープンソースから

Cloneを作成し（ローカルにコピーを作って）

それをアレンジして、自分（自社）用のアプリを構築するような場合：

ゼロからの設計ではなく、骨格部分が既にあるため、工期を短縮できる。
更新の履歴を調べることが容易。

複数人数での作業での、調整の効率化

Gitが難しい部分

ブランチができた際に、どの部分の修正がどの枝にあるかの把握が困難

ブランチを意識的に作成して、技術の試行錯誤に積極的に活用できれば、かなり強力なツールですが、この部分の壁が高い。

コマンドプロンプトを開く

Gnome端末を開き、
/home/WebDB/workspace/memopad
に移動する。

Change directoryコマンド

cd /home/WebDB/workspace/memopad

ここで、まず、git logと入力してみる。

まだ、リポジトリがない、というメッセージが返って来る。

```
[WebDB@cisnote memopad]$ pwd
/home/WebDB/workspace/memopad
[WebDB@cisnote memopad]$ git log
fatal: Not a git repository (or any of the parent directories): .git
[WebDB@cisnote memopad]$
```

Directoryの中身と、最初の応答

```
[WebDB@cisnote memopad]$ ls -Fal
合計 80
drwxrwxr-x 12 WebDB WebDB 4096  9月 13 11:10 2016 ./
drwxrwxr-x  4 WebDB WebDB 4096  9月 13 11:09 2016 ../
-rw-rw-r--  1 WebDB WebDB  543  9月 13 11:09 2016 .gitignore
-rw-rw-r--  1 WebDB WebDB  355  9月 13 11:10 2016 .project
-rw-rw-r--  1 WebDB WebDB 1727  9月 13 11:22 2016 Gemfile
-rw-rw-r--  1 WebDB WebDB 4460  9月 13 11:22 2016 Gemfile.lock
-rw-rw-r--  1 WebDB WebDB  374  9月 13 11:09 2016 README.md
-rw-rw-r--  1 WebDB WebDB  227  9月 13 11:09 2016 Rakefile
drwxrwxr-x 10 WebDB WebDB 4096  9月 13 11:09 2016 app/
drwxr-xr-x  2 WebDB WebDB 4096  9月 13 11:09 2016 bin/
drwxrwxr-x  5 WebDB WebDB 4096  9月 13 11:09 2016 config/
-rw-rw-r--  1 WebDB WebDB  130  9月 13 11:09 2016 config.ru
drwxrwxr-x  3 WebDB WebDB 4096  9月 13 11:30 2016 db/
drwxrwxr-x  4 WebDB WebDB 4096  9月 13 11:09 2016 lib/
drwxrwxr-x  2 WebDB WebDB 4096  9月 13 11:14 2016 log/
drwxrwxr-x  2 WebDB WebDB 4096  9月 13 11:09 2016 public/
drwxrwxr-x  8 WebDB WebDB 4096  9月 13 11:09 2016 test/
drwxrwxr-x  5 WebDB WebDB 4096  9月 13 11:14 2016 tmp/
drwxrwxr-x  3 WebDB WebDB 4096  9月 13 11:09 2016 vendor/
[WebDB@cisnote memopad]$ git log
fatal: Not a git repository (or any of the parent directories): .git
[WebDB@cisnote memopad]$
```

レポジトリの新規作成

/memopadのディレクトリで、以下のコマンドを実行する。

git init

Gitのリポジトリディレクトリ .gitが初期化される。

```
[WebDB@cisnote memopad]$ git init
Initialized empty Git repository in /home/WebDB/workspace/memopad/.git/
[WebDB@cisnote memopad]$
```

記録者の登録

今日学ぶのは、「一人で使う」モードですが、gitは複数の開発者が共同開発したり、公共の場にソースコードを提供する際に利用されるため、「誰」がリポジトリを更新したか、記録する必要があります。

このため、COMMITTERの氏名とメールアドレスを登録します。

```
[WebDB@cisnote memopad]$ git var GIT_COMMITTER_IDENT

*** Please tell me who you are.

Run

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

to set your account's default identity.
Omit --global to set the identity only in this repository.

fatal: unable to auto-detect email address (got 'WebDB@cisnote.(none)')
[WebDB@cisnote memopad]$
```

記録者の登録

--globalは省略しても構いません。「このPC環境を使うのは私だけ」という場合は、入れても特に問題はありません。

git config --global user.name “私の名前”

git config --global user.email xxx@xxx.xx

で設定します。

今現在の状態を登録する。

現在、memopadのディレクトリにいる場合、．(ピリオドーつ)は、current_directoryを意味します。

git add .

で、現在のディレクトリをgitに登録します。

この状態で、レポジトリへの登録を確定するため

git commit –m ‘初期状態’

と入力します。

現在の状態の登録

コミットが完了したら、git logで中身を見て下さい。

```
[WebDB@cisnote memopad]$ git add .
[WebDB@cisnote memopad]$ git commit -m '初期状態'
[master (root-commit) 0d22a12] 初期状態
 97 files changed, 1901 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 .project
 create mode 100644 Gemfile
 create mode 100644 Gemfile.lock
 create mode 100644 README.md
 create mode 100644 Rakefile
 create mode 100644 test/test_helper.rb
 create mode 100644 tmp/.keep
 create mode 100644 vendor/assets/javascripts/.keep
 create mode 100644 vendor/assets/stylesheets/.keep
[WebDB@cisnote memopad]$ git log
commit 0d22a12fa422dcf5a084a2e4203447747f56275c
Author: Ikuo Kobayashi <ikuo.kobayashi.t9@hosei.ac.jp>
Date:   Sat Sep 24 14:07:10 2016 +0900

    初期状態
[WebDB@cisnote memopad]$
```

.gitignoreについて

Gitの管理に含めないファイルを指定することができます。

ローカルに管理しているファイル、logファイル、tmp一時ファイル、データベースファイルなど

http://qiita.com/inabe49/items/16ee3d9d1ce68daa9fff

最初の「誤った」修正

先週は、「私のメモ帳」というタイトルだった部分を
「メモ帳のタイトル」という文字列に修正します。

app/config/locales/ja.yml

ファイルです。

この修正は、後から「この変更だけ」を元に戻す練習として使います。

辞書ファイル(yml)の復習

書式

シンボル名:(一つ以上の空白)コンテンツ

配列や、ハッシュ配列などで同じ深さの要素は、左側に同じ数の空白を入れる。

```
date:
  abbr_day_names:
  - Sun
  - Mon
  - Tue
  - Wed
  - Thu
  - Fri
  - Sat
```

前期からの違い(辞書の活用)

後期第1回で、「多国語化」を行いました。

このため、「準備中」などの文字列も、辞書から引いて来るようにしたため、

“準備中” → (t `not_ready’)

のように、「t(メソッド)」と、’not_ready’（見出し語）という組合せにし、その結果を()でまとめていきます。

最初の「誤った」修正


Gemfile
application_controller.rb
ja.yml
index.html.erb
ja:
memos_title: "メモ帳のタイトル"
Show: "表示"
index.html.erb
index.json.jbuilde
new.html.erb
show.html.erb
show.json.jbuilde
メモ帳のタイトル - Mozilla Firefox
メモ帳のタイトル
127.0.0.1:3000/memos
検索
Memos

修正をバックアップする。

git add –u

git commit –m ‘変更内容の見出し’

二つのコマンドで、修正をgitのレポジトリに記録する

-mのスイッチをつけずに、commitを行うと、viエディタが開かれて「編集」モードになる。

-mのスイッチでバックアップに「名前」をつける。

Viが苦手な人は、-mスイッチでコミット。

```
[root@cisnote memopad]# git add -u
[root@cisnote memopad]# git commit
[master 73b2d2b] ページタイトルを修正した。
 1 file changed, 1 insertion(+), 1 deletion(-)
[root@cisnote memopad]#
```

-mオプション：ありとなし

この後、-mなしでviモードに入った時の説明スライドが4ページ続きます。

git commit –m ‘変更内容の見出し’

で、直接コミットの内容のメモを指定した人は、4枚分スライドを読み飛ばしてください。

注：「変更内容の見出し」は、「変更内容の見出し」と入力するのではなく、実際の変更の内容をわかりやすく見出しにして記述して下さい。

Viの使い方(1)

Viエディタは、[挿入モード]と[コマンドモード]を切り換えて使います。

最初は、コマンドモードになっています。

i (英数字、小文字のi)を押すと、挿入モードになります。カーソルのある位置に、文字が挿入できます。

ここで、「ページタイトルを修正した。」などと、バックアップに名前をつけます。

挿入が終わったら、[ESC]を押して、コマンドモードに戻ります。

Viの使い方(2)

文字列の挿入が終わって、[ESC]を押したら、編集が済んだファイルを保存します。

: (英数字、半角の記号のコロン)を押すと、ファイルの保存や、編集ファイルの読み込みなどのコマンドを受け付けるモードになります。

w (英数字、半角小文字のw)を押すと、ファイルを上書き保存します。

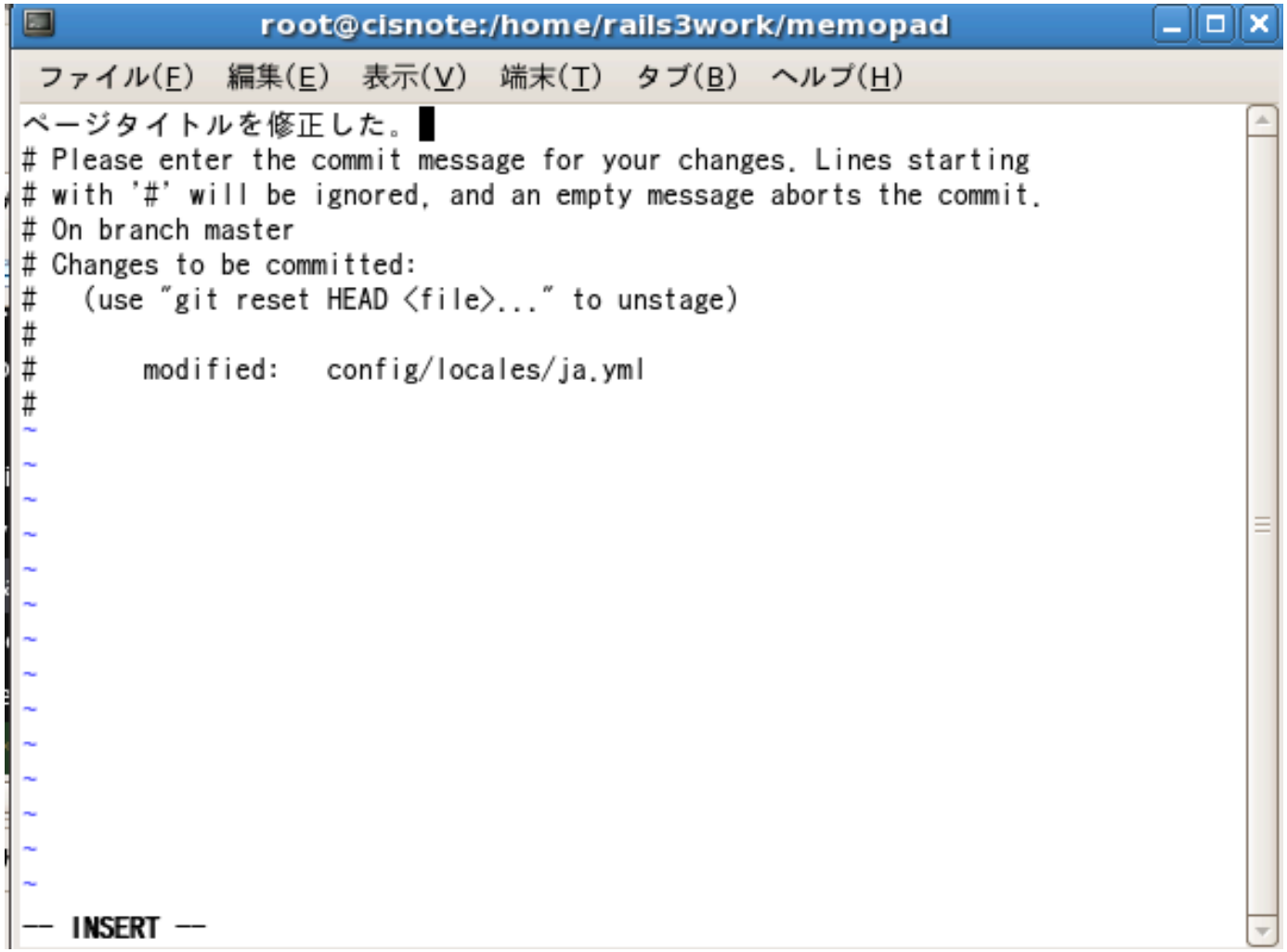
q (英数字、半角小文字のq)を押すと、終了します。

Viの利点

LINUXなどで、コマンドプロンプトしか使えない（サーバなどに障害が発生している）状況でも、ファイルを編集できます。

マウスを使わずに、キーボードだけで、すべての編集操作ができ、「3行コピー」が3yyとか、「貼付け」がpとか、数文字の入力だけで編集操作ができるため、慣れると高速にファイルを編集できます。

Commitのためのファイル編集画面



参考サイト

サルでもわかるGit入門

http://www.backlog.jp/git-guide/intro/intro1_1.html

画像もここから引用します。

Gitでの呼び名

Work tree

今現在、作業をしているgit管理下のディレクトリ
バックアップの対象であり、直接編集しているのは
このWork Tree上のファイル

サイトによっては、ワーキングツリーと書かれていたりします

リポジトリ

Gitでの版管理の記録／ローカルとリモートがある

リビジョン

版、とか「世代」

コミット

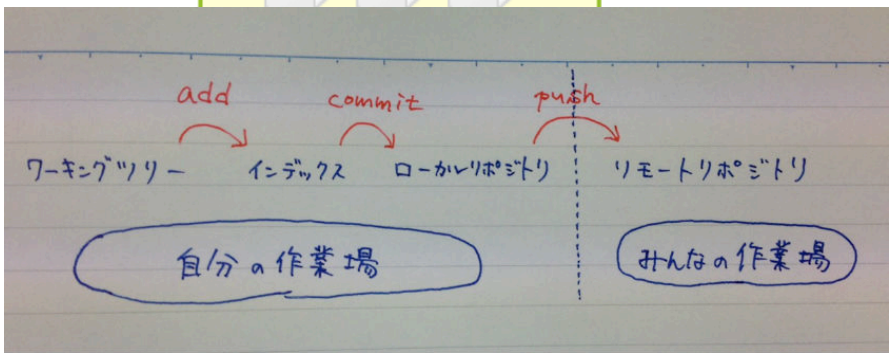
リポジトリに、新たにリビジョンを追加すること

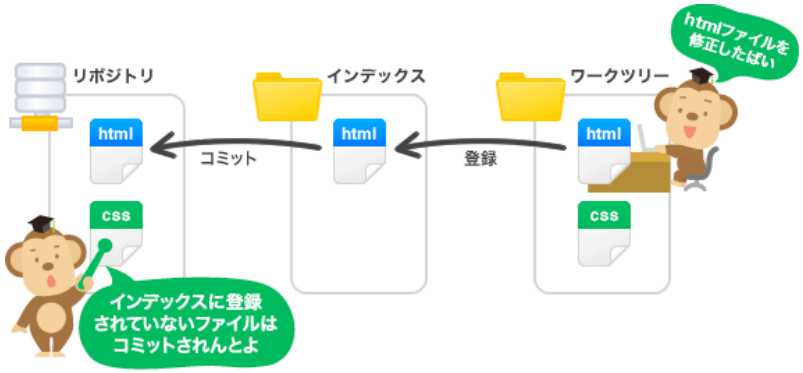
gitでの用語

http://qiita.com/shuntaro_tamura/items/db1aef9cf9d78db50ffe

インデックス(Index)
　次にcommitする状態を逐次記録したもの

コミット家系図
　リビジョンの関係を表した木構造





コミット前の、変更点確認

git diff

と入力すると、前回のコミット後にどんな修正がなされたか、どのファイルに追加や削除があったかを、表示させることができます。(IndexとWork treeの差分)

```
[root@cisnote memopad]# git diff
diff --git a/config/locales/ja.yml b/config/locales/ja.yml
index e5ae59d..3118b53 100644
--- a/config/locales/ja.yml
+++ b/config/locales/ja.yml
@@ -1,5 +1,5 @@
 ja:
-  memo_title: 私のメモ帳
+  memo_title: メモ帳のタイトル
   show: 表示
   edit: 編集
   destroy: 削除
[root@cisnote memopad]#
```

ここまでの小まとめ

gitの初期化（使い始め）

　git init　[プロジェクトのルートで行う。]

コミッタの登録（一度だけ、または、システムで一度）

　git config --global user.name “私の名前”

　git config --global user.email xxx@xxx.xx

保存対象のindexへの追加

　git add フォルダ名（またはファイル名）

これまでの保存対象の更新をindexに追加

　git add -u

保存（コミット）

　git commit -m “バックアップの名称”

Addとcommitを同時に行う

テストランの動作確認が終わったら、保存する。

git add –uを同時に行うcommitのオプションは、

git commit -a

で、-aオプションをつけると、git add –uのコマンドは不要になる。

さらに、-mオプションで名称も同時につける。

git commit -a -m “修正内容の見出し”

バックアップ対象を確認する

Eclipseのプロジェクトエクスプローラで見ると、
バックアップされる前の編集では、ファイル名やフォルダ名に[*]のマークが表示されている。

新たに new file を作成したとすると、その new fileは、バックアップされない。

理由： 最初のcommit時には、含まれていなかった。

git add app/hoge/newFile

で、new fileを「バックアップ対象」として追加し、コミットする。

git commit -m 'new fileを追加した。'

バックアップ記録の確認

git log　とコマンド入力する。

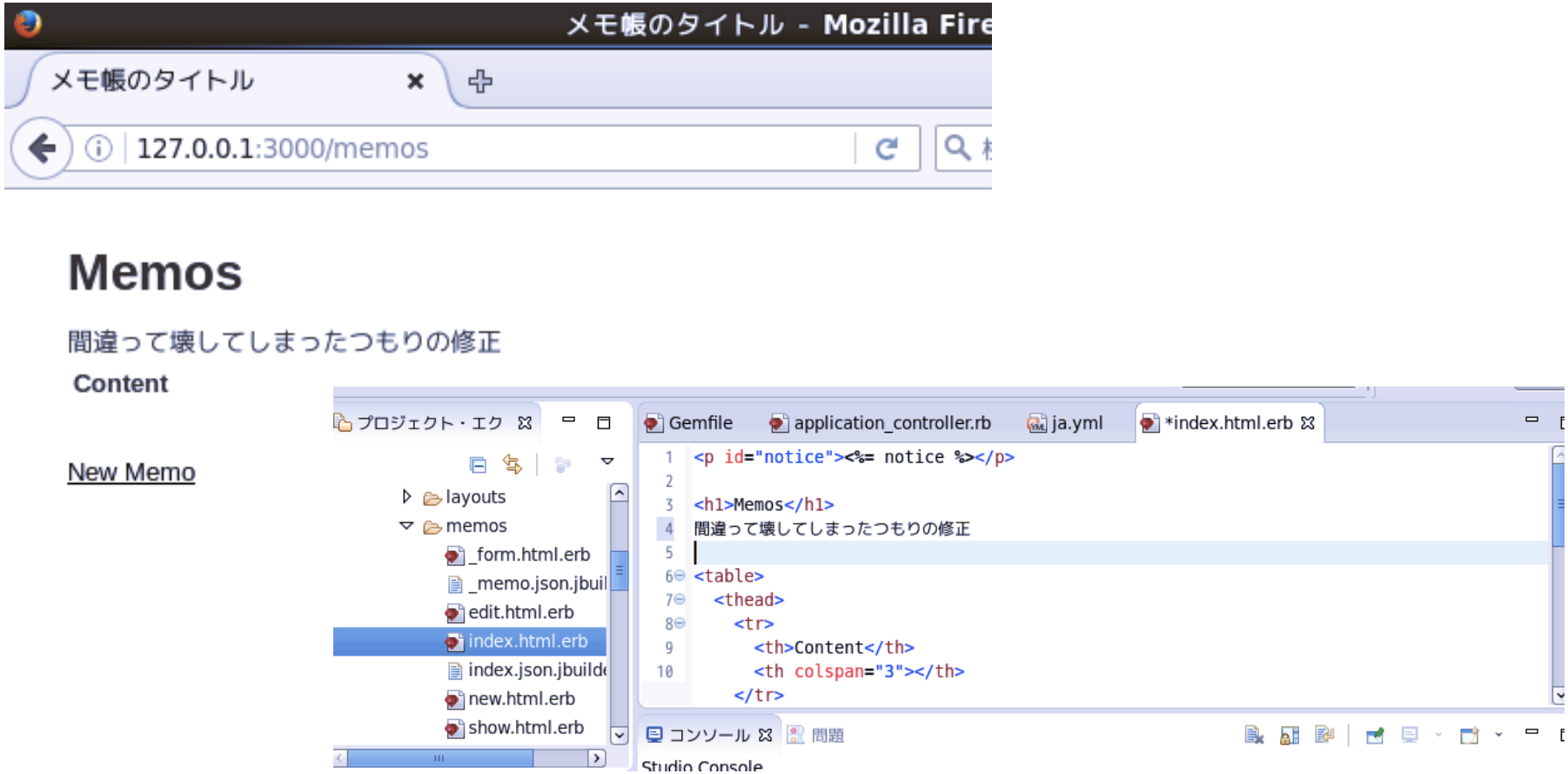
```
commit 3e37f6f6268f6ad8eb33485d8c5d4c05fd230961
Author: Ikuo Kobayashi <ikuo.kobayashi.t9@k.hosei.ac.jp>
Date:   Tue Oct 2 16:38:10 2012 +0900

    sharedを追加した。

commit 53449e5db295d9cfa544dcfbe0e85f1c5f0d7fac
Author: Ikuo Kobayashi <ikuo.kobayashi.t9@k.hosei.ac.jp>
Date:   Tue Oct 2 16:32:55 2012 +0900

    動作確認完了（画面分割）

commit e570786596ef784a4d55f1eaaa3569d68d7ef9ee
Author: Ikuo Kobayashi <ikuo.kobayashi.t9@k.hosei.ac.jp>
Date:   Tue Oct 2 15:05:14 2012 +0900

    画面を分割し、スタイルシートを編集した。

commit 73b2d2b97ebae4f92684b20fa4770a2ced6da344
Author: Ikuo Kobayashi <ikuo.kobayashi.t9@k.hosei.ac.jp>
Date:   Mon Oct 1 07:43:23 2012 +0900

    ページタイトルを修正した。

commit 36e34735c51f9c12ef34671f20632c1a7fb6ed16
Author: Ikuo Kobayashi <ikuo.kobayashi.t9@k.hosei.ac.jp>
Date:   Mon Oct 1 07:03:30 2012 +0900

    初期状態
:
```

最新のコミットを取り消す場合

最新のコミット内容が、間違っていたから、取り消したい場合の操作を行う。

余計な修正をして、ファイルを壊す

最後のコミットの後、余計な修正をしてファイルを壊したとする。



何をやってしまったか、確認する。

git commit -a -m '間違えた修正'

間違えた修正をコミットしてしまった。ログで見ると

git log

```
[WebDB@cisnote memopad]$ git log
commit 418d57e0efcf7fea66ae7890b638a703e193203b
Author: Ikuo Kobayashi <ikuo.kobayashi.t9@hosei.ac.jp>
Date:   Sat Sep 24 17:02:57 2016 +0900

    間違えた修正

commit 0ed7d0b5fb8b5c6571440585777bd83e1fa1d2b6
Author: Ikuo Kobayashi <ikuo.kobayashi.t9@hosei.ac.jp>
Date:   Sat Sep 24 15:53:37 2016 +0900

    タイトルの修正

commit 0d22a12fa422dcf5a084a2e4203447747f56275c
Author: Ikuo Kobayashi <ikuo.kobayashi.t9@hosei.ac.jp>
Date:   Sat Sep 24 14:07:10 2016 +0900

    初期状態
[WebDB@cisnote memopad]$
```

二通りの修正方法

HEADとIndexを元に戻す

git reset HEAD~

コミットした内容だけ戻し、ファイルは手作業で修正

HEAD, Indexと作業ファイルも元に戻す

git reset --hard HEAD~

プログラムも含めて戻すのはこちら。

証拠の隠滅・・・みたいな

練習時の注意

commitと、reset HEAD^の順番や組合せは、各自工夫して試して下さい。

有効なコミットを、reset HEAD^で取り消してしまっても、ファイル自体はcheckoutまで取り消されないので、コミットのやり直しができますが、「有効なバックアップ」を消すことがないように、それを覚えるための練習では、「消しても良い修正」を行って、コミットとやり直しを練習してみて下さい。

Git reset HEAD^

Gitのリポジトリだけ元に戻る。

```
[WebDB@cisnote memopad]$ git reset HEAD~
Unstaged changes after reset:
M       app/views/memos/index.html.erb
[WebDB@cisnote memopad]$ git log
commit 0ed7d0b5fb8b5c6571440585777bd83e1fa1d2b6
Author: Ikuo Kobayashi <ikuo.kobayashi.t9@hosei.ac.jp>
Date:   Sat Sep 24 15:53:37 2016 +0900

    タイトルの修正

commit 0d22a12fa422dcf5a084a2e4203447747f56275c
Author: Ikuo Kobayashi <ikuo.kobayashi.t9@hosei.ac.jp>
Date:   Sat Sep 24 14:07:10 2016

    初期状態
[WebDB@cisnote memopad]$
```

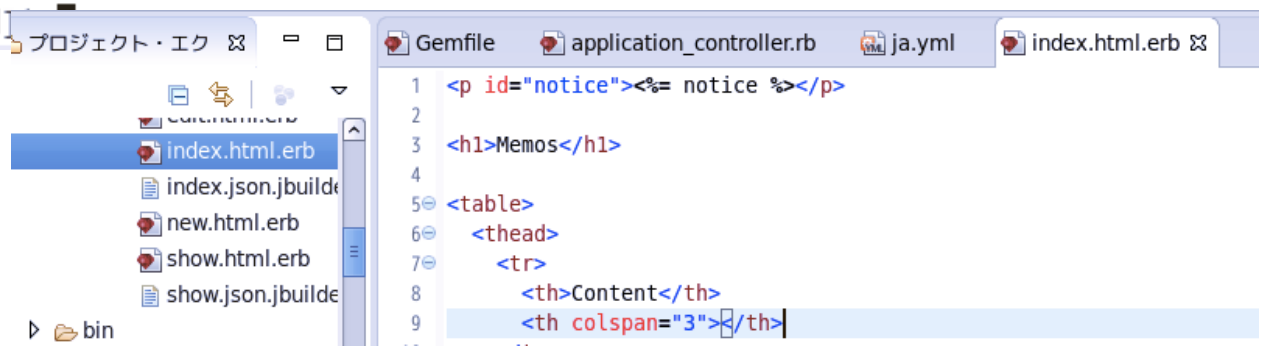
Gemfile | application_controller.rb | ja.yml | index.html.erb

```
1  <p id="notice"><%= notice %></p>
2
3  <h1>Memos</h1>
4  間違えて壊してしまったつもりの修正
5
6  <table>
7    <thead>
8      <tr>
9        <th>Content</th>
10       <th colspan="3"></th>
       </tr>
```

Git reset --hard HEAD^

ソースプログラムも元に戻る

```
[WebDB@cisnote memopad]$ git reset --hard HEAD~
HEAD is now at 0ed7d0b タイトルの修正
[WebDB@cisnote memopad]$ git log
commit 0ed7d0b5fb8b5c6571440585777bd83e1fa1d2b6
Author: Ikuo Kobayashi <ikuo.kobayashi.t9@hosei.ac.jp>
Date:   Sat Sep 24 15:53:37 2016 +0900

    タイトルの修正

commit 0d22a12fa422dcf5a084a2e4203447747f56275c
Author: Ikuo Kobayashi <ikuo.kobayashi.t9@hosei.ac.jp>
Date:   Sat Sep 24 14:07:10 2016 +0900

    初期状態
[WebDB@cisnote memopad]
```



Resetとrevertの違い

https://www.atlassian.com/ja/git/tutorial/undoing-changes#!overview

Revertとresetの違いに気をつけて下さい。

「取り消し」と「打ち消し」とで、用語を使い分けています。

HEAD~って何？

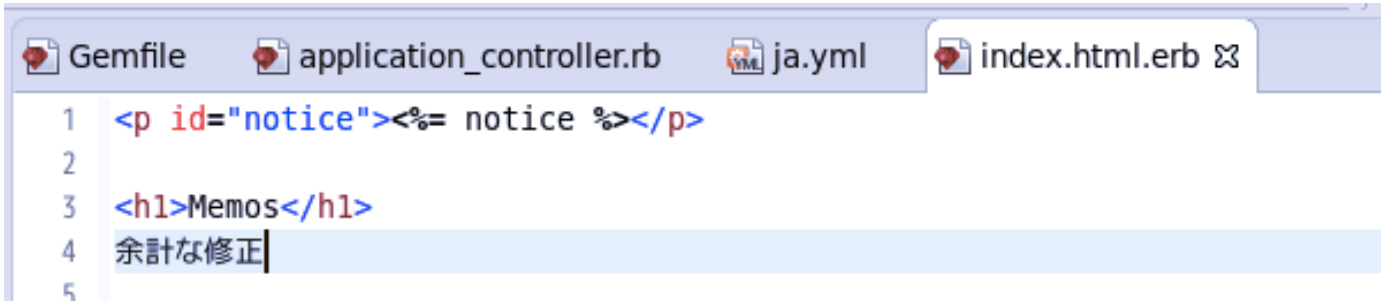
Git resetの後に来た HEAD~とは？
HEADとは、「現在最新のコミット」のこと

HEAD~は、最新の一世代前の親。
git reset HEAD~では、一世代前に戻す。

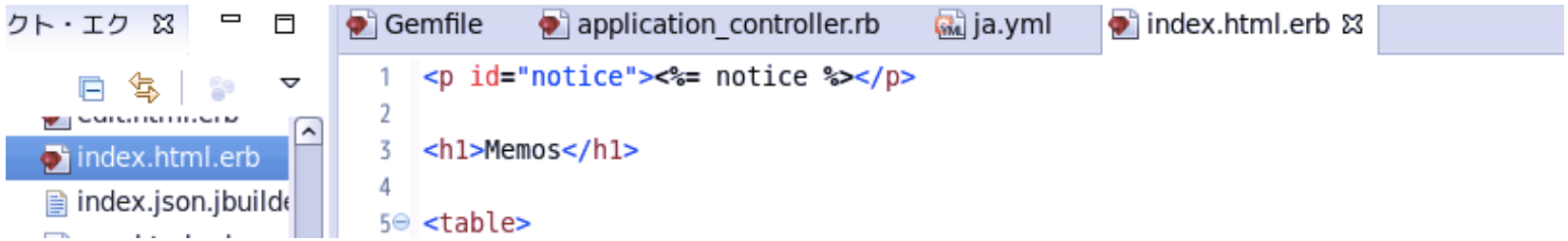
HEAD^と書かれているケースもあります。
HEAD^は、番号で指定(^は一番目の親)

最後のコミット後の修正取り消し

git reset --hard




```
[WebDB@cisnote memopad]$ git reset --hard
HEAD is now at 0ed7d0b タイトルの修正
[WebDB@cisnote memopad]$
```



checkout

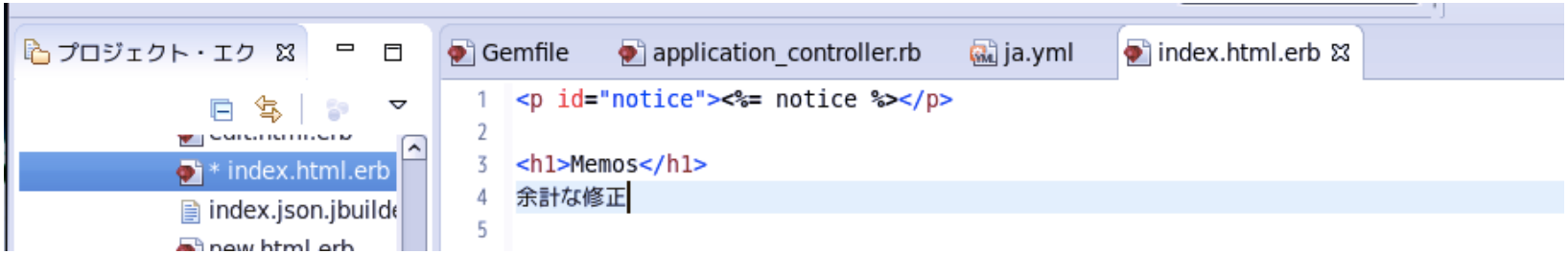
git checkout は、ファイルのチェックアウト、コミットのチェックアウト、ブランチのチェックアウトの3 つの異なる機能を有するコマンド

https://www.atlassian.com/ja/git/tutorial/undoing-changes#!checkout

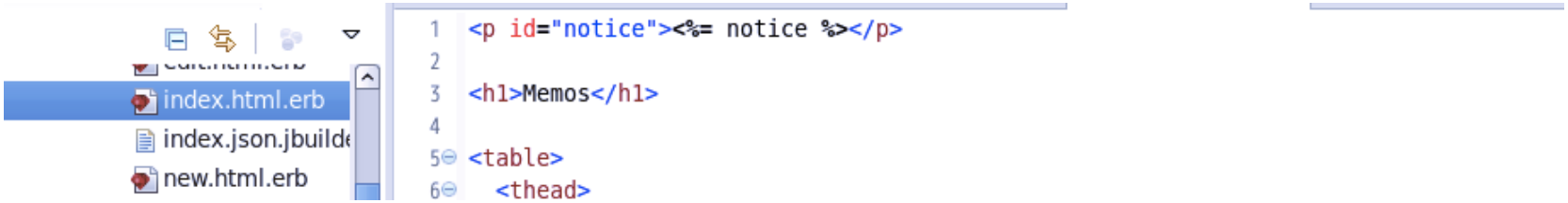
特定ファイルを特定コミットに戻す

git checkout <commit> ファイル名

と入力する。



Viコマンドでの編集になります。
[ESC]:wqとキー入力して書き込み終了させます。



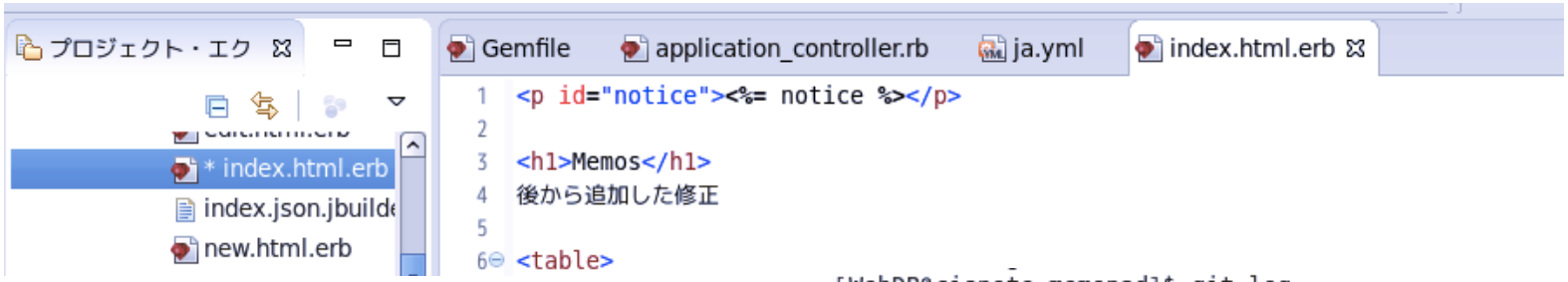
かなり古い修正のミスに気付いた

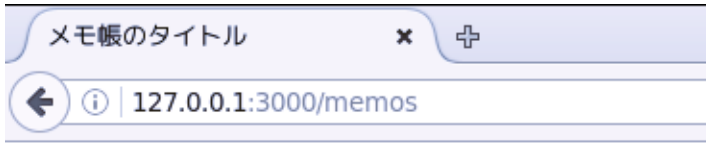
シナリオとして、「有効な修正」の前に、「うっかりミス」をして、それをバックアップしていた、とします。

見出し「私のメモ帳」を「メモ帳のタイトル」と書き直した修正が、有効な修正の前の「うっかりミス」だったとします。

後から修正（コミット）を追加

「誤った修正」を、少し前の「過去の修正」にするため、
新たに、「後から追加した修正」を加えます。





Memos

後から追加した修正

Content

New Memo

```
[WebDB@cisnote memopad]$ git log
commit c72a385d06d607c1809a8cde1d2bd1f9f897f90c
Author: Ikuo Kobayashi <ikuo.kobayashi.t9@hosei.ac.jp>
Date:   Sat Sep 24 17:44:51 2016 +0900

    後から追加した修正のコミット

commit 0ed7d0b5fb8b5c6571440585777bd83e1fa1d2b6
Author: Ikuo Kobayashi <ikuo.kobayashi.t9@hosei.ac.jp>
Date:   Sat Sep 24 15:53:37 2016 +0900

    タイトルの修正

commit 0d22a12fa422dcf5a084a2e4203447747f56275c
Author: Ikuo Kobayashi <ikuo.kobayashi.t9@hosei.ac.jp>
Date:   Sat Sep 24 14:07:10 2016 +0900

    初期状態
[WebDB@cisnote memopad]$
```

最初の「誤った」修正（再掲）


Gemfile
application_controller.rb
ja.yml
index.html.erb
ja:
memos_title: "メモ帳のタイトル"
Show: "表示"
index.html.erb
index.json.jbuilde
new.html.erb
show.html.erb
show.json.jbuilde
メモ帳のタイトル - Mozilla Firefox
メモ帳のタイトル
127.0.0.1:3000/memos
検索
Memos

コミットの番号を調べる。

git log

と入力してみる。

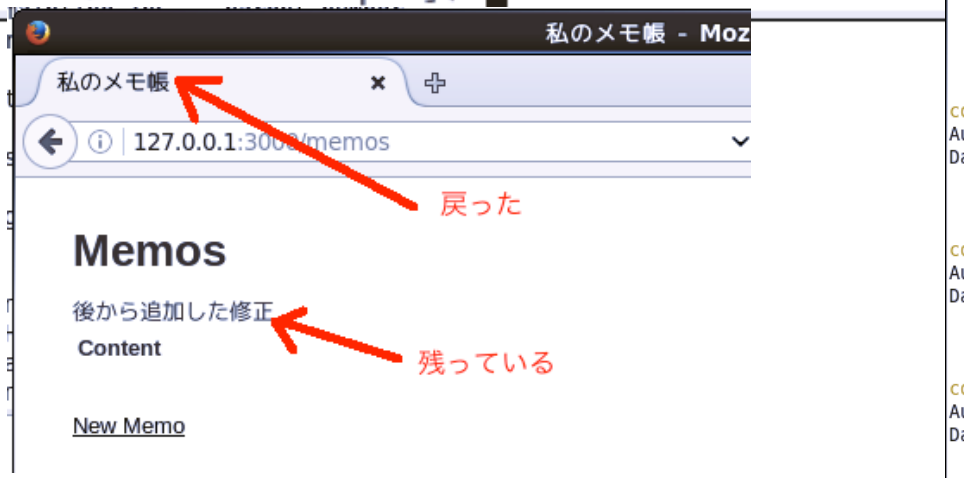
```
[WebDB@cisnote memopad]$ git log
commit c72a385d06d607c1809a8cde1d2bd1f9f897f90c
Author: Ikuo Kobayashi <ikuo.kobayashi.t9@hosei.ac.jp>
Date:   Sat Sep 24 17:44:51 2016 +0900

    後から追加した修正のコミット

commit 0ed7d0b5fb8b5c6571440585777bd83e1fa1d2b6
Author: Ikuo Kobayashi <ikuo.kobayashi.t9@hosei.ac.jp>
Date:   Sat Sep 24 15:53:37 2016 +0900

    タイトルの修正

commit 0d22a12fa422dcf5a084a2e4203447747f56275c
Author: Ikuo Kobayashi <ikuo.kobayashi.t9@hosei.ac.jp>
Date:   Sat Sep 24 14:07:10 2016 +0900

    初期状態
[WebDB@cisnote memopad]$
```

特定のコミットを取り消す

git revert <コミット番号>

これによって、「過去の特定のコミット」を取り消した、新しい「コミット」が作成されます。（記録は残っています。）

```
[WebDB@cisnote memopad]$ git revert 0ed7d0b5fb8b5c6571440585777bd83e1fa1d2b6
[detached HEAD 835625d] Revert "タイトルの修正"
 1 file changed, 1 insertion(+), 1 deletion(-)
[WebDB@cisnote memopad]$
```




```
[WebDB@cisnote memopad]$ git log
commit 835625dbecf5705aef4ee0ad7df2e427b53b6fed
Author: Ikuo Kobayashi <ikuo.kobayashi.t9@hosei.ac.jp>
Date:   Sat Sep 24 17:57:57 2016 +0900

    Revert "タイトルの修正"

    This reverts commit 0ed7d0b5fb8b5c6571440585777bd83e1fa1d2b6.

commit c72a385d06d607c1809a8cde1d2bd1f9f897f90c
Author: Ikuo Kobayashi <ikuo.kobayashi.t9@hosei.ac.jp>
Date:   Sat Sep 24 17:44:51 2016 +0900

    後から追加した修正のコミット

commit 0ed7d0b5fb8b5c6571440585777bd83e1fa1d2b6
Author: Ikuo Kobayashi <ikuo.kobayashi.t9@hosei.ac.jp>
Date:   Sat Sep 24 15:53:37 2016 +0900

    タイトルの修正

commit 0d22a12fa422dcf5a084a2e4203447747f56275c
Author: Ikuo Kobayashi <ikuo.kobayashi.t9@hosei.ac.jp>
Date:   Sat Sep 24 14:07:10 2016 +0900

    初期状態
[WebDB@cisnote memopad]$
```

ブランチを作るやり方

世代を遡って

2世代前の修正にミスがあったので、これを消したい！
（各自の世代については、各自の状況に応じて読み替えて下さい。）

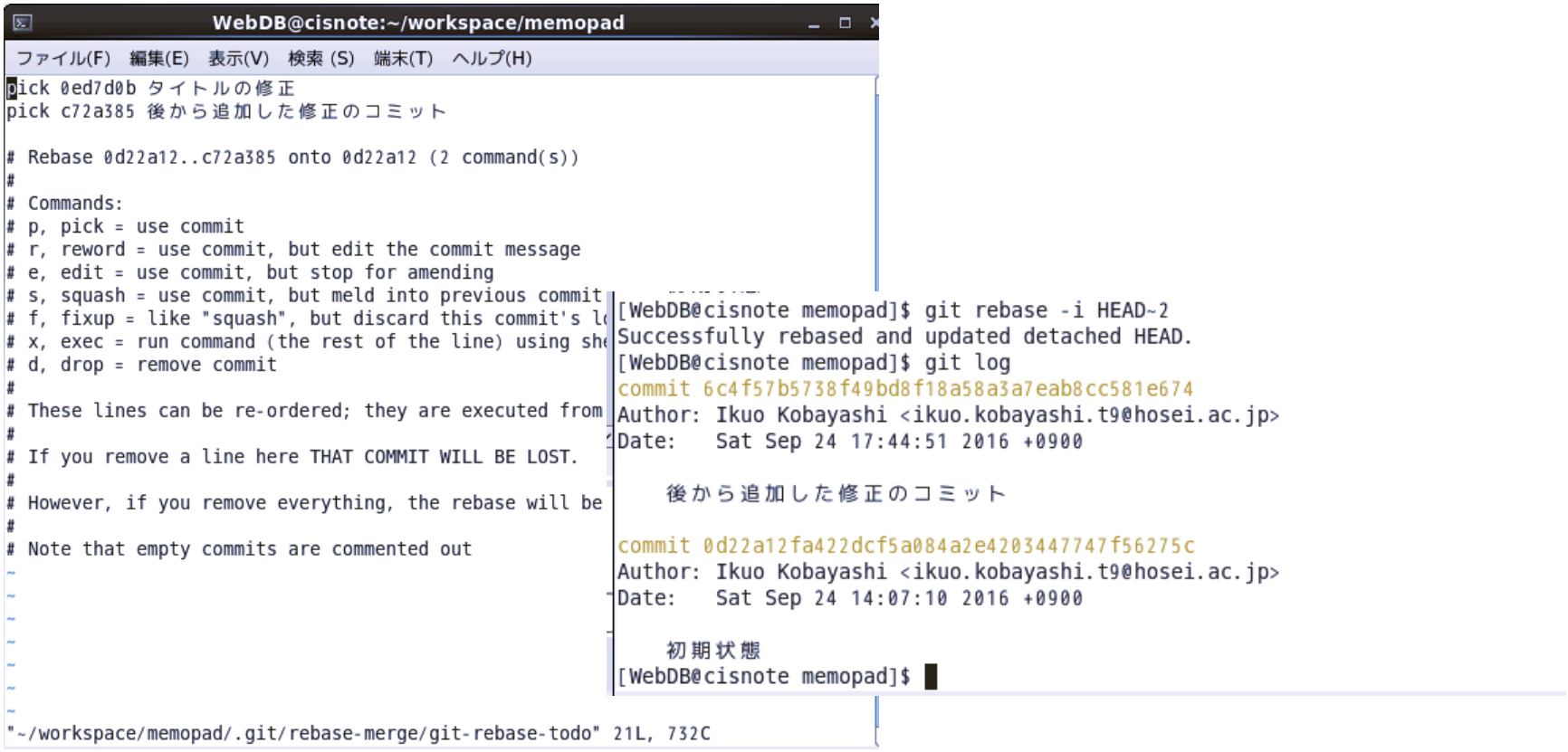
2世代前まで遡る。

```
git rebase -i HEAD~2
```

で、2世代前に戻る。→Viが開きます。

Viで編集：指定する。

タイトルの修正（誤った修正）のコミットを、
Pickからdropに書き換える。(指示に従って下さい)



過去の消滅

こちらの場合には、過去が消える。

```
[WebDB@cisnote memopad]$ git rebase -i HEAD~2
Successfully rebased and updated detached HEAD.
[WebDB@cisnote memopad]$ git log
commit 6c4f57b5738f49bd8f18a58a3a7eab8cc581e674
Author: Ikuo Kobayashi <ikuo.kobayashi.t9@hosei.ac.jp>
Date:   Sat Sep 24 17:44:51 2016 +0900

    後から追加した修正のコミット

commit 0d22a12fa422dcf5a084a2e4203447747f56275c
Author: Ikuo Kobayashi <ikuo.kobayashi.t9@hosei.ac.jp>
Date:   Sat Sep 24 14:07:10 2016 +0900

    初期状態
[WebDB@cisnote memopad]$
```

「サルでもわかる？」

どの「コミット」がどんな内容を持っているか、コマンドの意味や、結果は、慣れるまで大変だと思います。

できるだけ多くの「図」を参照して、ファイルに何が起きているか、チェックしながら、トレースして下さい。

「サルでもわかる」サイトを見ても、慣れるまでは難しいと思います。

今日の課題レポート（提出）

C/B評価課題シナリオ

※　Gitを用いてBackupを作成します。

※　その後、ついうっかり(?)、プロジェクトのappの下をそっくり削除してしまったことに気付きました。

※　Gitのコマンドを用いて、バックアップを作成した時点のプロジェクトに、そっくり復元するための手順を説明し、実際に削除して復活させて、動作確認を行った際に必要な画面を報告して下さい。

（報告の内容でC/B評価いずれか判断します。）

LINUXで中身のあるディレクトリを丸ごと消すコマンド

rm –fr ディレクトリ名

rmはremoveです。

オプションの-fは、force (強制的に)、

-rは、recursively (再帰的に)つまり、下のディレクトリ構造も含めて、という意味です。

今日の課題–A評価シナリオ

以下の流れで文書編集を辿り、課題を行って下さい。
(シナリオ説明：P1/3)

※「法政大学の西館213教室で、10月9日午後1時から、Microsoft社の Bill Gates氏が講演会を行う」という、学生向けのアナウンス文書 announce.txtを作成した。 → git保存

※ その後、正しくは、Macrosoft社のMs. Kate Hillsであり、講演会ではなく講習会であると、判明したため、文書を修正した。 → git保存

A/S評価シナリオ （2/3）

※　その後、さらに関係者から日付も間違えていると聞いて、10月16日（日）に修正した。 → git保存

※　さらに、講演内容として「Moon Micro Systems社の開発環境 Moon-1を用いた開発」というテーマを追記し、学生の受講料2，000円という情報も追記して、保存した。 → git保存

※ ところが、やはり講演者はMicrosoft社の Bill Gates氏であると教えられ、この分の修正を元に戻すため、講演者修正の作業だけ、取り消すことにした。

A/S評価 レポート課題

4回のgit保存のリビジョンが残されているものとして、最後の「講演者修正」の部分は、gitのコマンド操作だけで行う場合に、どんな手順で、どんなコマンドを用いれば良いか、時間の流れに沿って説明して下さい。(A評価)

この時の、それぞれのコマンドの意味を説明し、どんな効果(機能)があるのか、解説して下さい。(S評価)

本日の欠席課題

レポート課題と同じです。

B / C評価課題だけ出来ていれば、出席に切り替えます。

次回予告

次回は、入力の不備を指摘するValidationの使い方を学びます。