Mercurial Queues ハンズオン

TokyoMercurial#4.5

自己紹介

飯野卓見 a.k.a. @troter

株式会社タイムインターメディア所属

Mercurialエバンジェリスト見習い

#TokyoMercurialの開催

#scmbc Mercurialサポートスタッフ

#pyfes Mercurialハンズオンの開催

アジェンダ

第一部 導入

- Mercurial Queues(MQ)
- パッチキュー
- パッチキューの操作(=MQの使い方)

第二部 ハンズオン

- 準備編
- 基本編
- 履歴改変編
- 発展編、へのポインタ

Mercurial Queues

Mercurial Queuesとは

- Mercurialに同梱されているExtension
  - MQExtension
  - hgrcに書けばすぐに利用可能
- 変更内容をパッチで管理
  - リポジトリにコミットするわけではない
  - パッチの集合 = パッチキュー
  - パッチ↔リビジョン(コミット)相互変換できる
- 前身はquilt
  - (Linux)カーネル開発のためのユーティリティ
    - パッチ管理ツールを用いた開発元のソースコードの修正

MQでは何ができるのか

- コミットの削除
- コミット順序の変更
- コミット内容の変更
  - コミットログの変更
  - 変更内容の変更
- コミットの結合
  - コミットの分割は工夫次第
- コミット場所の変更
  - ブランチの移動（rebase）

使い所

- コミットログ間違えてた
- 新規作成したファイルを追加し忘れた
- 仕掛りの作業を横に避けておきたい
- コミットするブランチを間違えた
- rebaseしたい
- 複数に分けたコミットを一つにしたい
- TDDでRed→Greenなコミット順序にしたい

MQ使いこなすとどうなる？

- 残念なコミットが減る
  - コミットログを何度でも書き直せる
  - 変更内容がより論理単位になる
- 同期ためのマージコミットが減る
  - コミットの移動、rebaseが簡単にできる
  - リビジョングラフが自由自在
- 仕掛りの作業が怖くなくなる
  - 途中までやってパッチにする
  - 別の作業するパッチを前に持ってくる
  - そもそもパッチキューを切り替える

最終的には

MQが無いMercurial = 残念な子

という印象になります

パッチキュー

パッチキューとは

- MQが取り扱うパッチの集合のこと
  - MQを使う=パッチキューを操作する
- パッチキューには次が含まれる
  - パッチ
  - パッチの順序
  - パッチの適応状態
- パッチキューは複数持つことができる
  - hg qqueue、詳細はハンズオンの発展編で
- パッチキューのバージョン管理もできる
  - --mqオプション、詳細は(ry

パッチキューのディレクトリ構成

リポジトリルート/

.hg/

patches/　デフォルトのパッチキュー

series　パッチ一覧

status　パッチの適用状況

<patch-file1> パッチ1

<patch-file2> パッチ2

...

ところでパッチとは

- 変更内容が記載されたファイル
- diffの出力で、patchの入力

```
# パッチファイルの作成
hg diff > hoge.diff
#パッチファイルの適用
patch -p1 < hoge.diff
```

パッチの例 1(hg exportの出力)

```
# HG changeset patch
# User Matt Mackall <mpm@selenic.com>
# Date 1337369673 18000
# Branch stable
# Node ID 6476a21337a6942ee3b4b06ceb93f2902775e2b1
# Parent  c20efe04cd7a265dc3b0f28ac56a88f5d9af8859
osutil: handle deletion race with readdir/stat (issue3463)

diff --git a/mercurial/osutil.c b/mercurial/osutil.c
--- a/mercurial/osutil.c
+++ b/mercurial/osutil.c
@@ -331,6 +331,9 @@
                    err = lstat(fullpath, &st);
 #endif
                    if (err == -1) {
+                           /* race with file deletion? */
+                           if (errno == ENOENT)
+                                   continue;
                            strncpy(fullpath + pathlen + 1, ent->d_name,
                                    PATH_MAX - pathlen);
                            fullpath[PATH_MAX] = 0;
```

パッチの例 2(hg exportの出力)

```
# HG changeset patch
# User Matt Mackall <mpm@selenic.com>
# Date 1337186433 18000
# Node ID 46c15dff3497a0d8e6603cf8920061237b87612b
# Parent  99f369f5a8dbc192cd520aece8437f3182b1ac50
largefiles: fix default clone destination regression

diff --git a/hgext/largefiles/overrides.py b/hgext/largefiles/overrides.py
--- a/hgext/largefiles/overrides.py
+++ b/hgext/largefiles/overrides.py
@@ -705,7 +705,7 @@

 def overrideclone(orig, ui, source, dest=None, **opts):
     if dest is None:
-        dest = defaultdest(source)
+        dest = hg.defaultdest(source)
     if opts.get('all_largefiles') and not hg.islocal(dest):
             raise util.Abort(_(
             '--all-largefiles is incompatible with non-local destination %s' %
```

パッチキューの操作

パッチキューの操作

- 新しいパッチの作成(qnew)
- パッチの適用、非適用(qpush, qpop)
- パッチの適応状態の確認(qseries)
- 変更内容の確認(qdiff)
- パッチの更新(qrefresh)
- パッチの順序入れ替え(qpush --move)
- パッチの結合(qfold)
- パッチ↔リビジョン(コミット)相互変換(qimport, qfinish)

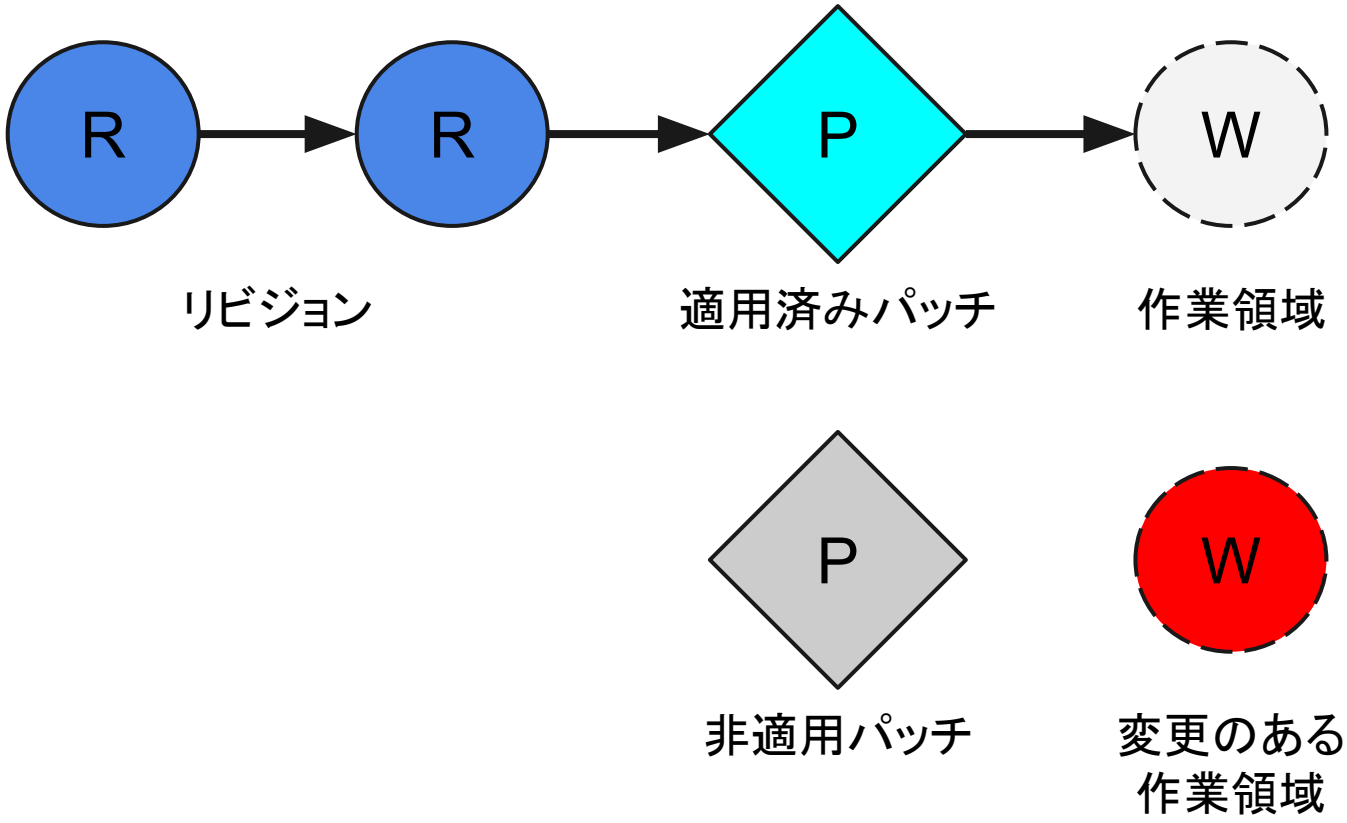
注意:いくつかの危険な操作

パッチを更新する操作は取り消せません
具体的には次のコマンドです。

qrefresh
qfold

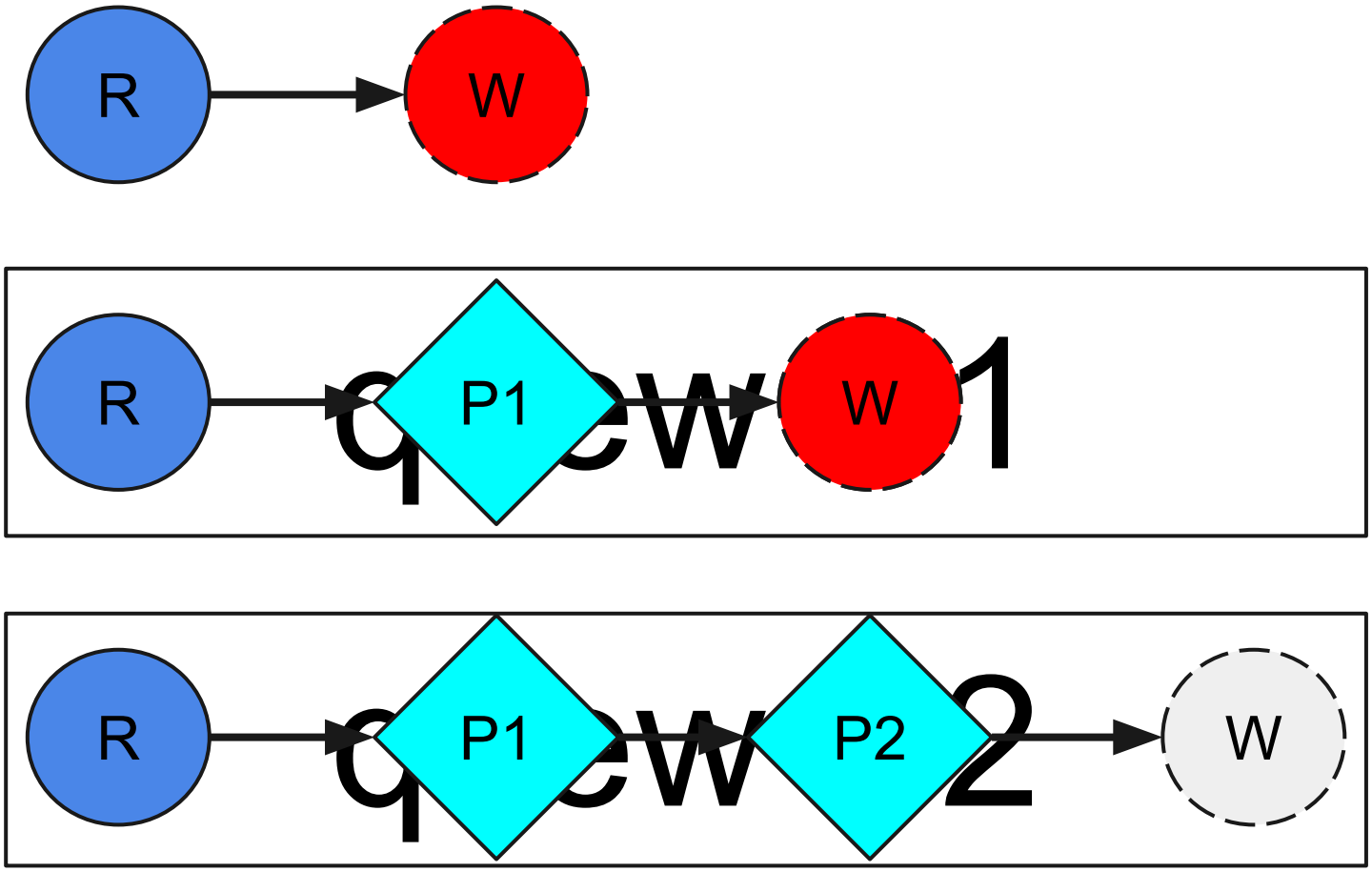
どうしても取り消したい場合は、パッチキュー自体をバージョン管理する必要があります。

ちなみに僕はバージョン管理してません。

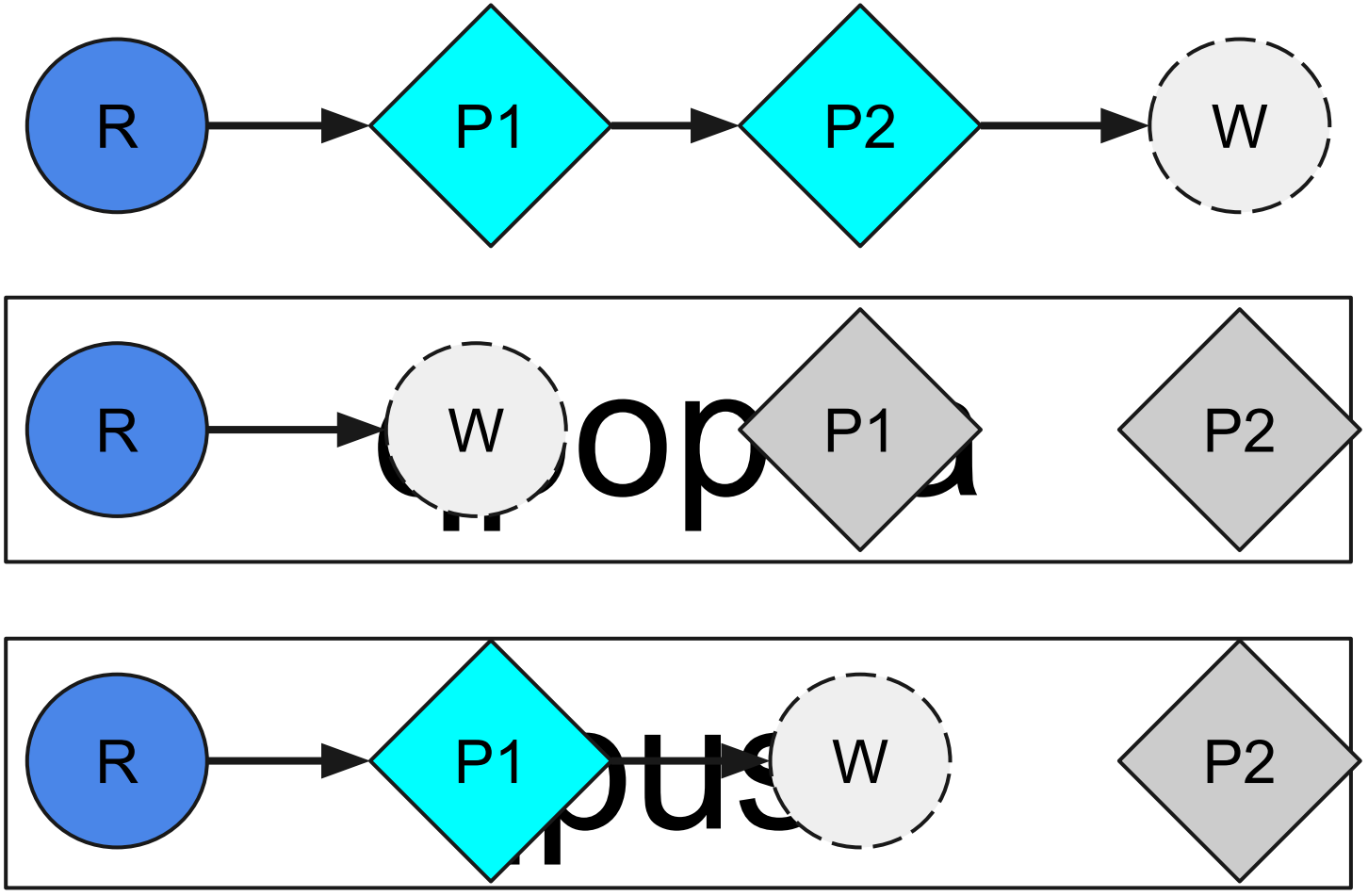


グラフの凡例

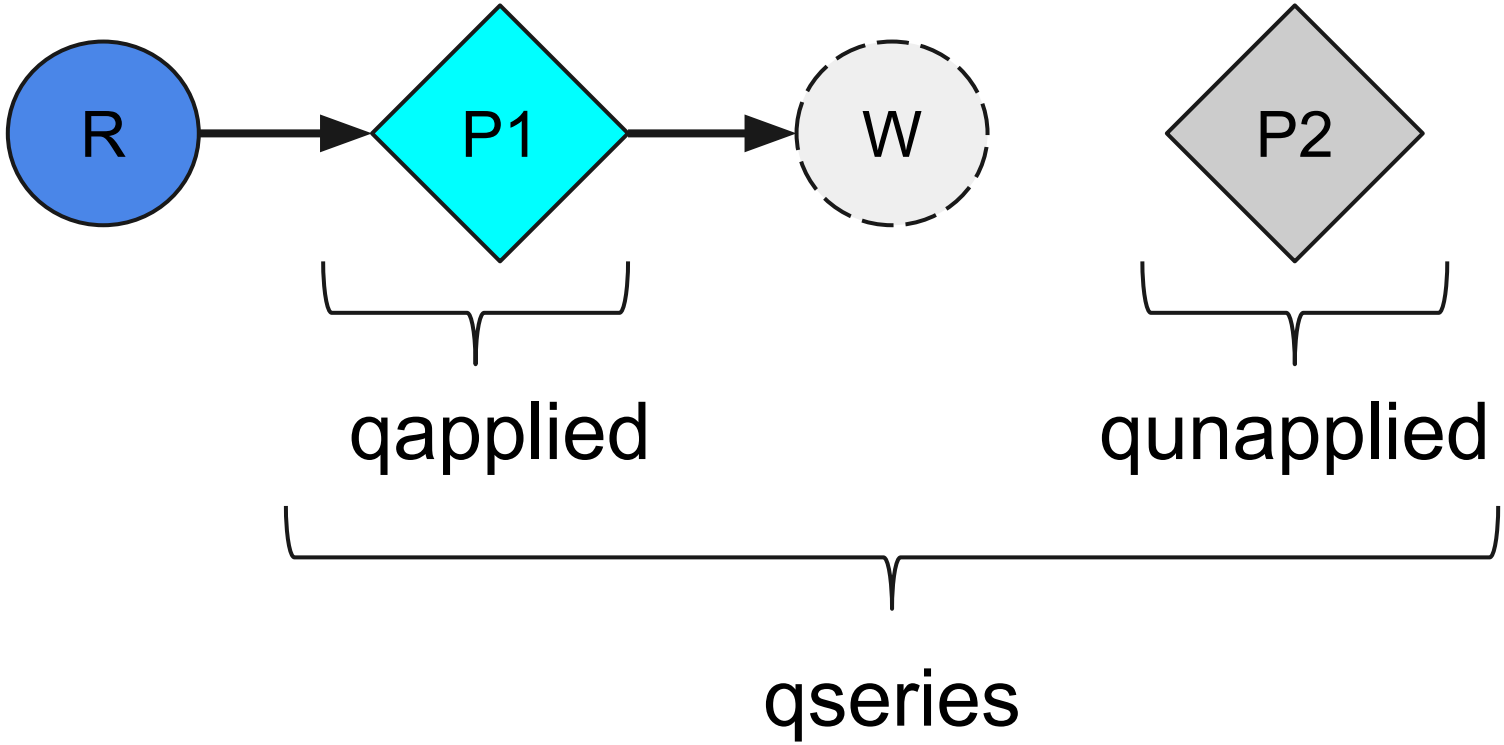
新しいパッチの作成



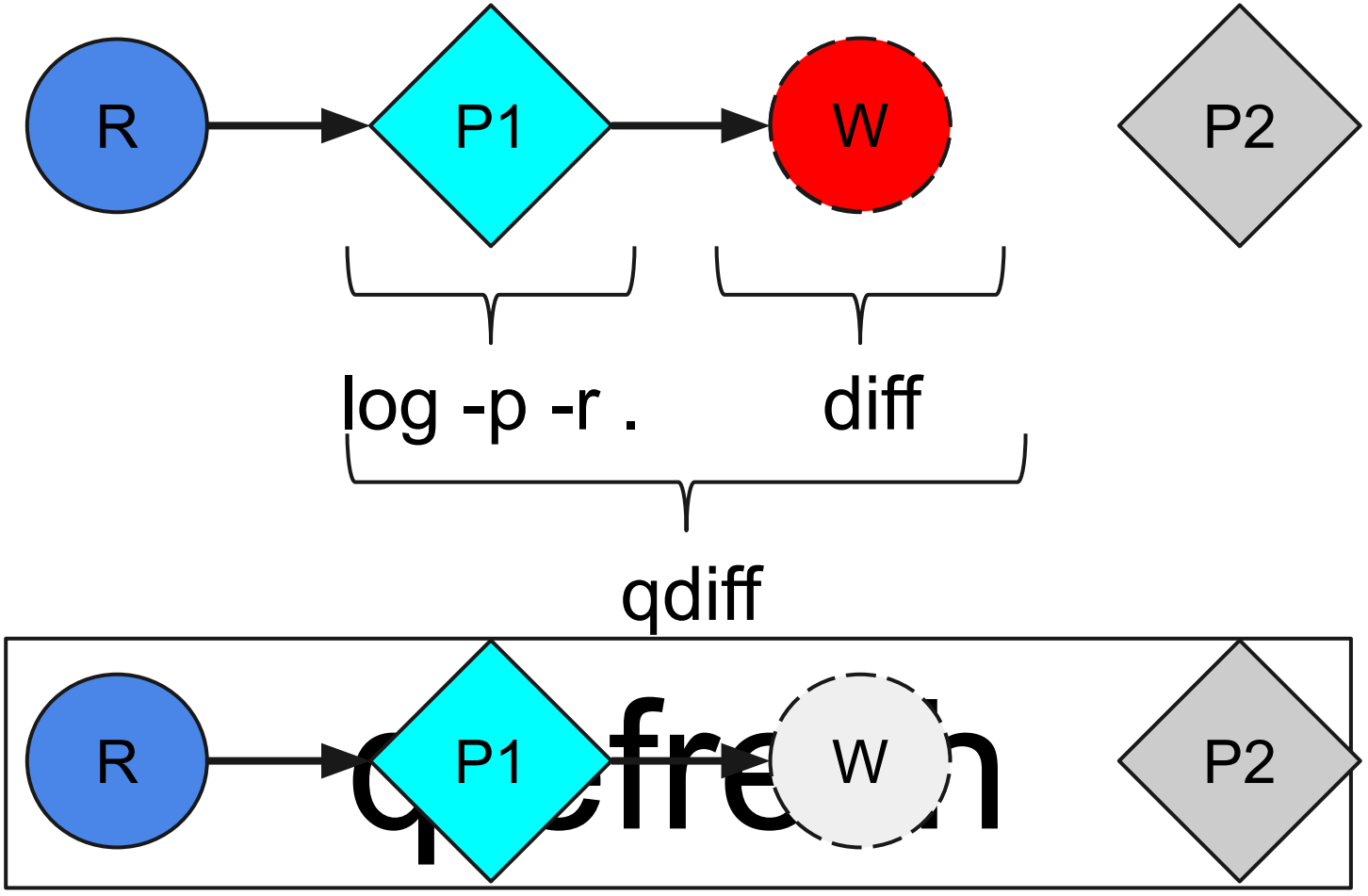
パッチの適用、非適用



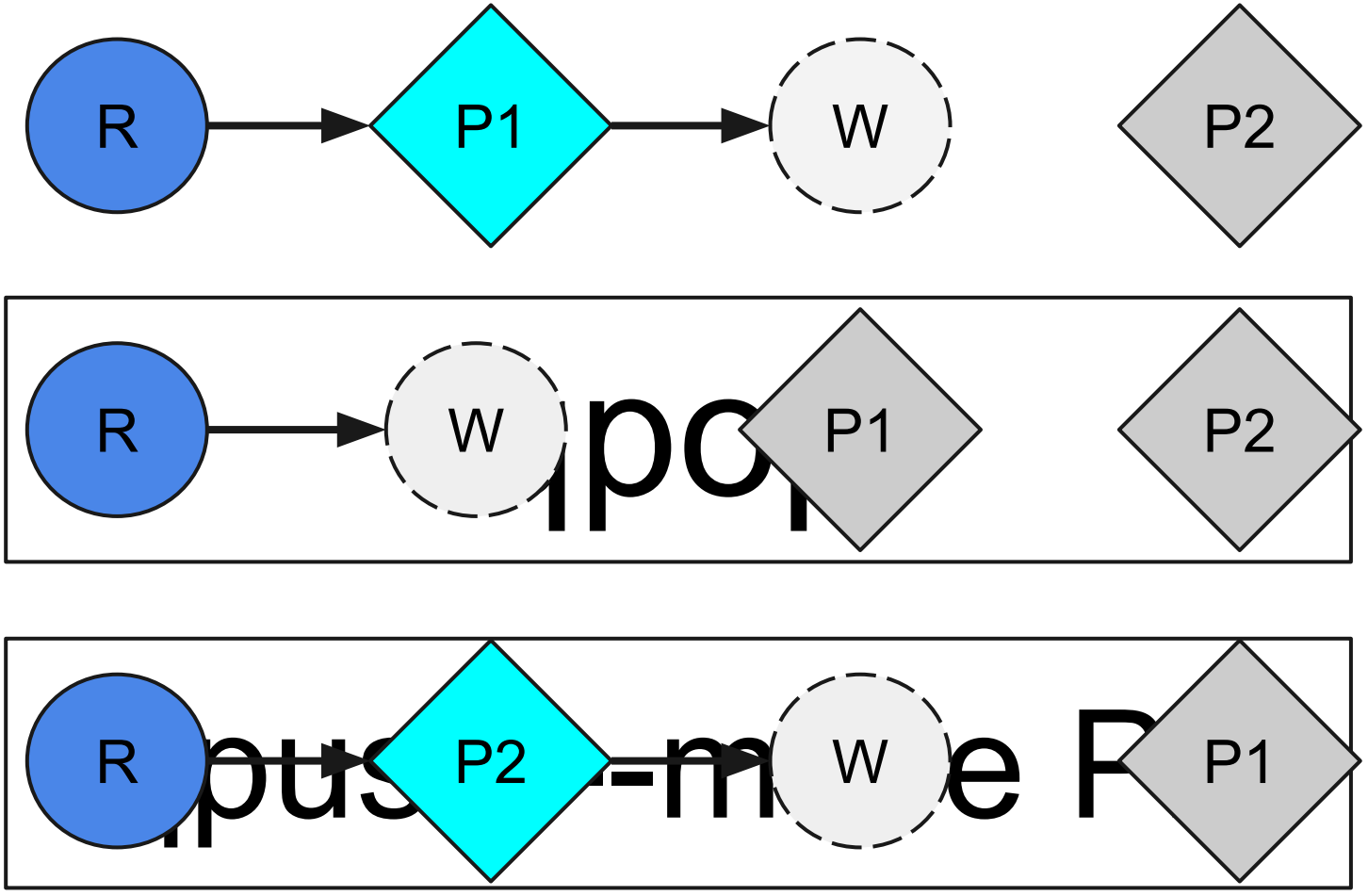
パッチの適用状態の確認



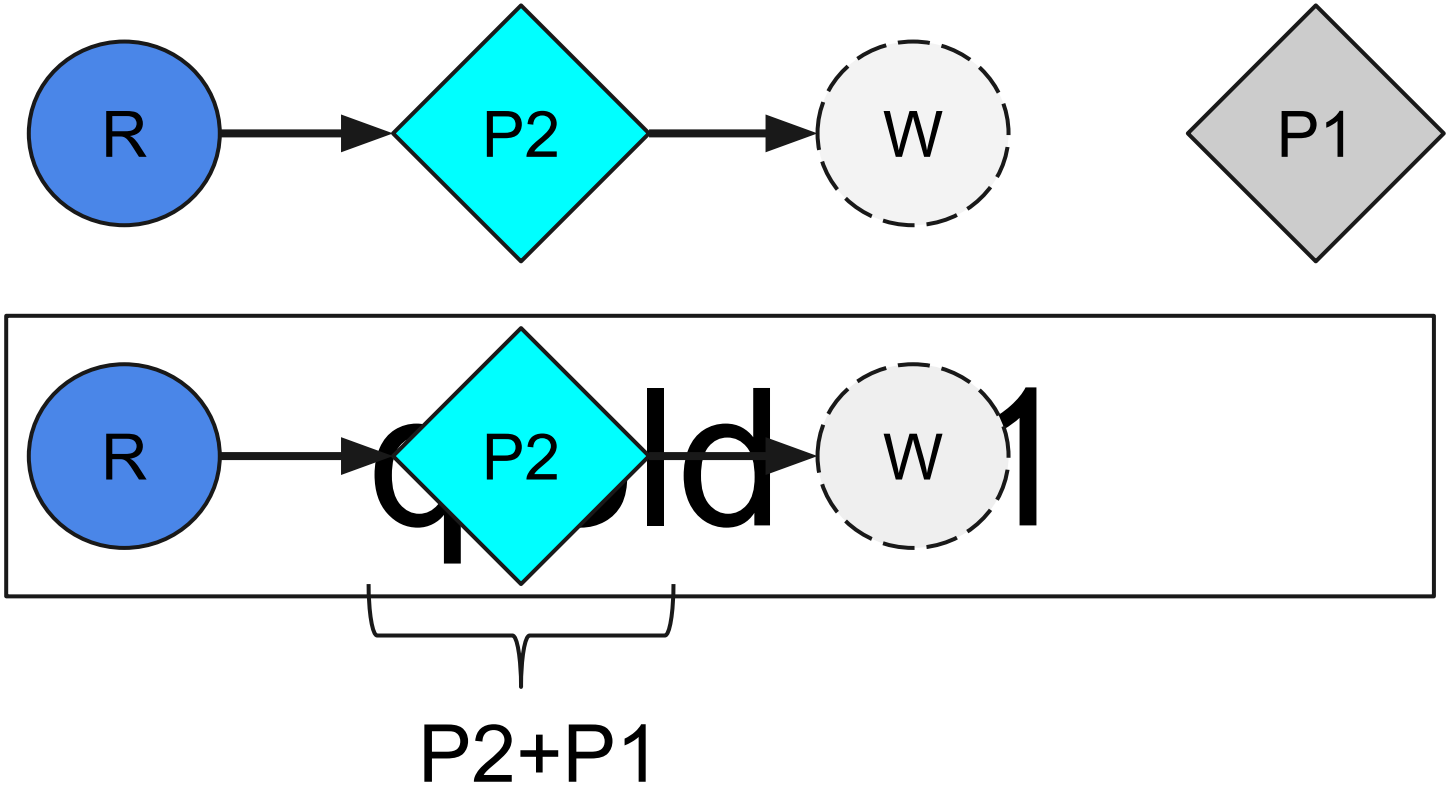
パッチの更新



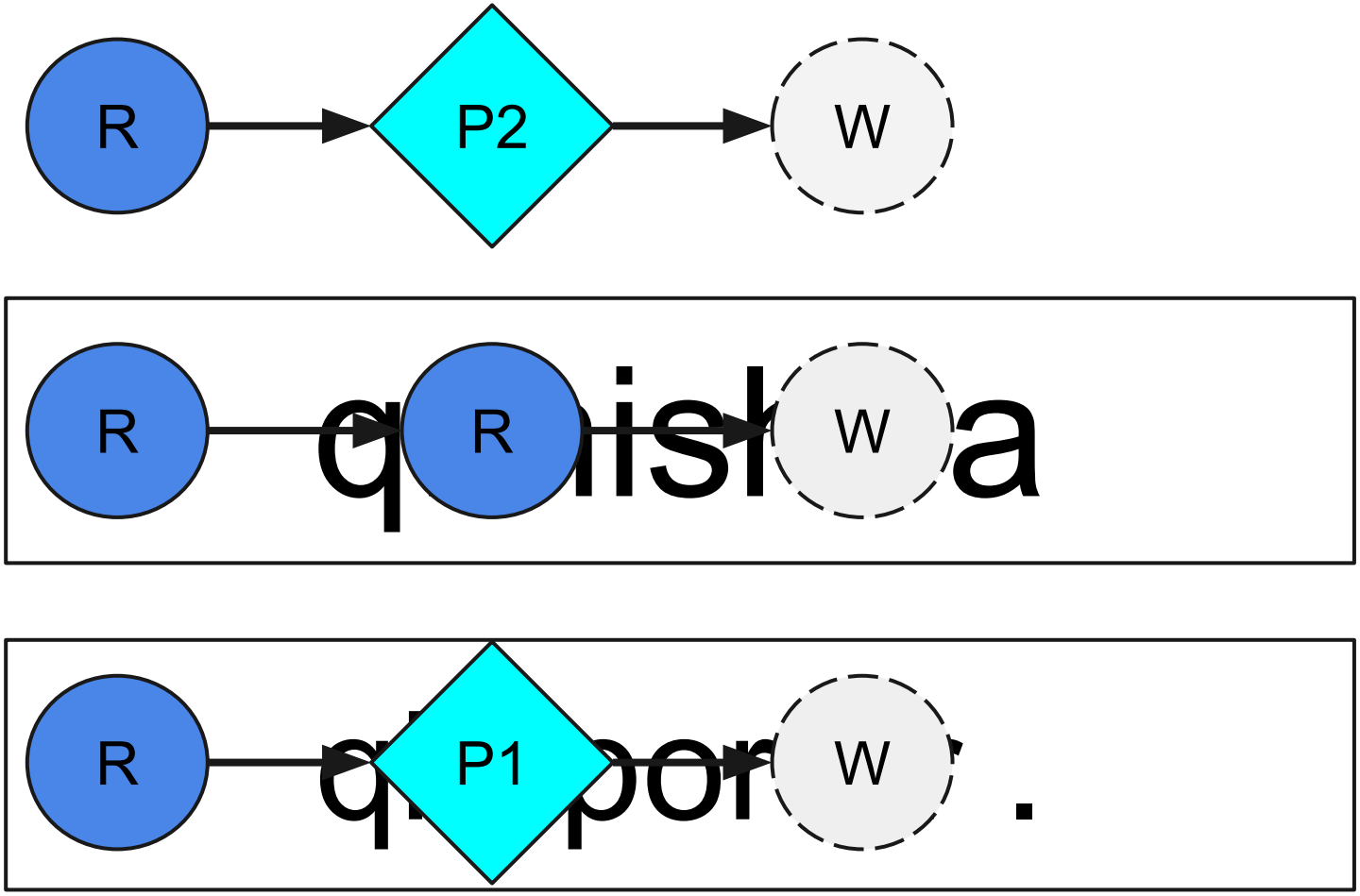
パッチの順序入れ替え



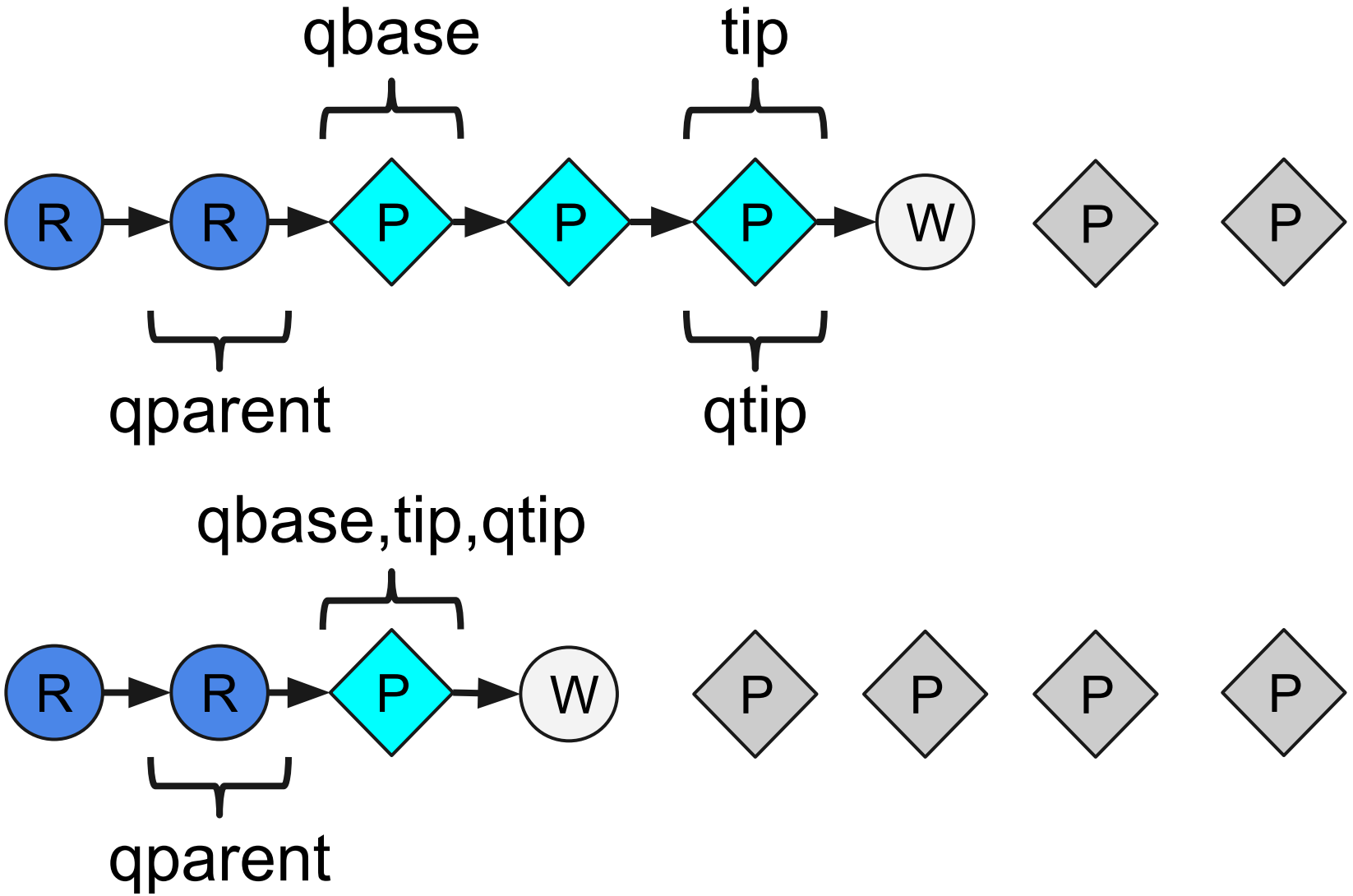
パッチの結合



パッチ↔リビジョン相互変換



パッチキュー？


qbase
tip
R
R
P
P
P
W
P
P
qparent
qtip
qbase,tip,qtip
R
R
P
W
P
P
P
P
qparent

MQ用のタグ

Mercurial Queues まとめ

MQを使いこなす事によってMercurialの

封印された真の能力

を引き出すことができる。

なぜ封印(= 禁術指定)されているか？
それは、常人が扱うには危険すぎるため。

ソフトウェア開発という戦場を生き抜き、メンバーを救うためにはMQという禁術を使いこなす他無い。

これで概要説明は終わりです

10分休憩です。

休憩後はハンズオンです。

ハンズオン

準備編

ハンズオンを始める前に次の項目を確認

1. コンフリクトの解決できますか？
2. 名前付きブランチのマージできますか？
3. 転んでも泣かないですか？
4. Mercurialのバージョンはいくつですか？

確認後、MQのための設定を行います

Mercurialのバージョンの確認

2.x系をインストールしてきましたか？

```
hg version
```

2.x系でない人は次を実行！

```
easy_install -U mercurial
pip install -U mercurial
```

MQの設定

次の設定をhgrcに追記してください。

```
[extensions]
mq =
[diff]
git = True
[mq]
secret = True
```

ハンズオン課題の取得

課題のリポジトリです。

- https://bitbucket.org/troter/tokyomercurial-4.5-mq-handson/overview

ZIPファイルをダウンロードします。

- https://bitbucket.org/troter/tokyomercurial-4.5-mq-handson/get/default.zip
- http://goo.gl/lv0r1

READMEがあるので読んでね！

課題の内容

tokyomercurial-4.5-mq-handson/

- basic/ 基礎編
  - mq.rst
- rewrite/ 履歴改変編
  - commitlog.sh
  - qpush-conflicts.sh
  - ...
- advance/ 発展編、のはずだったが力尽きた
  - ...

ハンズオン基礎編

基礎編

まずは基本のコマンドの操作に慣れましょう

basicディレクトリのmq.rstの内容をmqを使いながら編集していきます。

```
cd basic
hg init
hg ci -A -m "initial commit"
```

qnew

mq.rstの内容を充実させていきます。次のことに注意してください。

1. qrefreshの項まで作業してください。
2. サブコマンドごとにqnewしてください。
3. qnew前にかならずdiff, qdiffで内容を確認してください。
4. qnew後はqseries --summaryでパッチの状態を確認してください。

qrefresh

コミットログや変更内容を更新する場合はqrefreshを利用します。

1. 現在のパッチのコミットログを変更してください（qrefresh -e）
2. ファイルに変更を加え、パッチの変更内容を更新してください（qrefresh）

qpop & qpush & qgoto

今qseriesするとパッチがたくさん表示されると思います。

1. qpop, qpush, qgotoを使ってパッチの適用状態を変更してみましょう。
2. コマンドの実行前と実行後で、作業領域の内容とqseriesの出力結果がどう異なるか、確認してみましょう。

.hg/patchesの確認

.hg/patchesがパッチキューの本体です。

1. パッチキューにどんなファイルがあるか確認しましょう
2. ファイルの内容も覗いてみましょう

※ファイルを編集しないよう気をつけて下さい。

qfinish -a

作成したパッチをリビジョンに変換してみましょう。

次のコマンドで適用中のパッチをすべてリビジョンに変換しましょう。

```
hg qfinish -a
```

残りのサブコマンド

残りのサブコマンドについてもmqを駆使して編集していってください。

30分ほど試行錯誤してみてください。

基礎編は終わりです

10分休憩です。

ハンズオン履歴改変編

履歴改変編

rewriteディレクトリに課題があります。

```
cd rewrite
```

シェルスクリプトを実行すると課題のリポジトリと課題の内容が表示されます。

では順番に課題を解いていきましょう。

コミットログの改変

次のスクリプトを実行してください

```
./commitlog.sh
```

ディレクトリに移動して課題を解いてください

```
cd commitlog
```

qpush時のコンフリクト解決

次のスクリプトを実行してください

```
./qpush-conflicts.sh
```

ディレクトリに移動して課題を解いてください

```
cd qpush-conflicts
```

注意：qpushはマージしない

qpushでパッチを適用する際に、マージアルゴリズムは働きません。

MQだけで完結させるのではなく、graftやrebaseと組み合わせるとマージアルゴリズムが利用出来て便利です。その場合は一旦qfinishでリビジョンに変換します。

rebase

2つの課題があります。

次のスクリプトを実行してください

```
./rebase-1.sh
./rebase-2.sh
```

それぞれ課題を解いてください

パッチの順序入れ替え

2つの課題があります。

次のスクリプトを実行してください

```
./reorder-1.sh
./reorder-2.sh
```

それぞれ課題を解いてください

パッチの結合

2つの課題があります。

次のスクリプトを実行してください

```
./folding-1.sh
./folding-2.sh
```

それぞれ課題を解いてください

パッチの分割（難易度高）

2つの課題があります。

次のスクリプトを実行してください

```
./split-1.sh
./split-2.sh
```

それぞれ課題を解いてください

履歴改変編は終わりです

10分休憩です。

ハンズオン発展編

発展編、へのポインタ

- MQと別のコマンドの連携
  - recordエクステンション
  - rebaseエクステンション
  - graft
- パッチキューを複数持つ
  - qqueue
- パッチキューのバージョン管理
  - --mqオプション
- 条件付きパッチ適用
  - qguard, qselect

MQと別のコマンドの連携

recordエクステンション

mqとrecordがともに有効になっている場合、qrecordコマンドが利用出来ます。recordのインターフェースでqnew出来ます。

rebaseエクステンション

mqを使った場合は、マージをしてくれないので、複雑なコミットを移動させる場合はrebaseを使うと便利です。

MQと別のコマンドの連携

graft

mqではコミットを複製するわけではありません。チェリーピックする場合はgraftを利用する必要が有ります。また、graft+stripを組み合わせる事によりrebaseと同じ事が出来ます。

パッチキューを複数持つ

次のエントリの前半を参照してください。。

- Mercurial Advent Calendar 2011 24日目 hg qqueueや--mqを使ってパッチキューを活用する - 放牧日記

パッチキューのバージョン管理

次のエントリの後半を参照してください。。

- Mercurial Advent Calendar 2011 24日目 hg qqueueや--mqを使ってパッチキューを活用する - 放牧日記

条件付きパッチの適用

うまい参考URLがなかった。

Mercurial: The Definitive Guide
の日本語訳ですかね。

参考文献