本ドキュメントでは，Xcrypt に同梱されたファイルステージング機能モジュール file_stager.pm の使用方法について説明する．

第1章 ファイルステージング外部仕様

1.1. 概要

ファイルステージング外部仕様の概要を説明する。

1.1.1. 利用条件

下記の２つのコマンドがログインノード、計算ノードの両方にインストールされている事。

- zip コマンドのインストール

/usr/bin にインストールされている事。

- unzip コマンドのインストール

/usr/bin にインストールされている事。

1.1.2. ファイルステージング使用のための記述

パッケージモジュール名は以下の通りである。

- file_stager

ファイルステージング機能を Xcrypt で使用したい時はユーザスクリプトのパッケージモジュールに関する記述部分に'file_stager'を記述しなければならない。

ユーザスクリプトの記述例を以下に示す。

```
use base qw(file_stager core);
```

1.1.3. ユーザスクリプトのジョブに書ける事

以下の表にユーザスクリプトに書けるハッシュキーと指定するものを示す。

指定できるハッシュキー	指定するもの
JS_stage_in_files	ステージインファイル
JS_stage_out_files	ステージアウトファイル
JS_staging_base_dir	ステージングベースディレクトリ

以下にユーザスクリプトへの記述例を以下に示す。

- ステージイン、ステージアウト

```
'JS_stage_in_files' => [ 'input' ],
'JS_stage_out_files' => [ 'output' ]
```

- ベースディレクトリ

```
'JS_staging_base_dir' => "$ENV{HOME}/cvs/xcrypt/xcrypt-hg/work",
```

1.1.4. ベースディレクトリ

ファイルステージングにおける相対パス指定の基準となるディレクトリをベースディレクトリと呼び、定義することができる。定義する場所は xcr ファイル(ユーザスクリプト)である。ベースディレクトリのデフォルトはワークディレクトリである。ファイルステージングにおけるベースディレクトリの位置付けとしては以下の通りである。

- ステージインにおける転送先ディレクトリ
- ステージアウトにおける転送元ディレクトリ

また xcr ファイルにベースディレクトリを定義した例である。

```
use base qw(file_stager core);

%template = (
    'id' => "job0",
    'exe0@' => sub{"touch work/foo$VALUE[0].txt"},
    'RANGE0' => [0..1],
    'JS_stage_out_files@' => sub{[sub{"foo$VALUE[0].txt"},
sub{"work/tmp/foo2$VALUE[0].txt"}]},
    'after_in_job' => 'print "after_in_job¥n";',
    'after' => sub {
                    if(-f "work/tmp/foo2$VALUE[0].txt"){
                    print "exist¥n";}
    },
    'JS_staging_base_dir' => "$ENV{HOME}/cvs/xcrypt/xcrypt-hg/work",
);

my @jobs = &prepare_submit_ sync(%template);
```

1.2. システム設定

ベースディレクトリをシステム管理者が予め定義しておくことができる。

定義場所は config ファイルのジョブスケジューラ.pm である。

xcr ファイルと config ファイルの両方にベースディレクトリの定義が記述されていた場合は xcr ファイルの定義が優先される。config ファイルの sh.pm にベースディレクトリを定義する場合の例である。

```
# Config file for sh
use config_common;

$jsconfig::jobsched_config{"sh"} = {
    # commands
    qsub_command => "$ENV{XCRYPT}/lib/config/run-output-pid.sh",
    qdel_command => "kill -9",
    qstat_command => "ps",
    # standard options
    jobscript_preamble => ['#!/bin/sh'],
```

```
    jobscript_workdir => sub { '.'; },
    qsub_option_stdout => workdir_file_option('-o ', 'stdout'),
    qsub_option_stderr => workdir_file_option('-e ', 'stderr'),
    # ステージングベースディレクトリ
    staging_basedir => "/home/xcryptuser/mount/",
    extract_req_id_from_qsub_output => sub {
        my (@lines) = @_;
        if ($lines[0] =~ /([0-9]*)/) {
            return $1;
        } else {
            return -1;
        }
    },
    extract_req_ids_from_qstat_output => sub {
        my (@lines) = @_;
        my @ids = ();
        foreach (@lines) {
            if ($_ =~ /^¥s*([0-9]+)/) {
                push (@ids, $1);
            }
        }
        return @ids;
    },
};
```

1.3. 拡張モジュールのファイルステージング

拡張モジュールを作成するにあたって計算ノード側で処理を行いたいとする。この場合、計算ノードで実行する処理をスクリプトにし、ジョブスクリプトにそのスクリプトを実行する文を差し込む必要がある。そしてジョブが実行される時、ジョブスクリプトの文から拡張モジュールのスクリプトが呼ばれ処理が実行される。ファイル共有時であれば特に問題は無いのだが、ファイル非共有時に拡張モジュールのスクリプトが計算ノード側にないという問題が起こる。

その解決策として file_stager ではユーザ指定のファイルステージングの他に拡張モジュール用のファイルステージング方法を用意する。

1.3.1. 拡張モジュールのステージイン

拡張モジュールのステージインについて説明する。拡張モジュールのステージインでは計算ノードで行いたい拡張モジュールスクリプトの転送を補助する。以下の記述で拡張モジュールスクリプトをステージインできる。

```
sub before{
    my $self = shift;
    push(@{$self->{xcr_stage_in_files_list}}, “extension_module.sh”);
}
```

- @{$self->{xcr_stage_in_files_list}}

ステージインするファイルを追加していく配列。定義は file_stager のコンストラクタで行う。

- extension_module.sh

拡張モジュールが計算ノードで実行したい処理スクリプト。名前は一例であるのでそれぞれの名前で構わない。

1.3.2. 拡張モジュールのステージアウト

拡張モジュールのステージアウトについて説明する。拡張モジュールのステージアウトでは計算ノードで行われた処理に対する出力結果ファイルの回収を補助する。以下の記述で拡張モジュールに必要な出力結果ファイルをステージアウトできる。

```
sub before{
    my $self- = shift;
    push(@{$self->{xcr_stage_out_files_list}}, “extension_module.out”);
}
```

- @{$self->{xcr_stage_out_files_list}}

ステージアウトするファイルを追加していく配列。定義は file_stager のコンストラクタで行う。

- extension_module.out

拡張モジュールが計算ノードで行った処理に対する出力結果ファイル。名前は一例であるのでそれぞれの名前で構わない。

1.4. ステージング不可なファイル

ファイルステージングに指定不可能なファイル名がある。以下にその種類を示す。

- ファイル名に‘,’(カンマ)カンマが入っているファイル
- ファイル名に‘　’(空白)が入っているファイル
- ファイル名に‘*’(アスタリスク)が入っているファイル
- ファイル名に‘?’(クエスチョンマーク)が入っているファイル

ファイル名に以上の記号が含まれているとプログラムが異常動作する可能性がある。ファイルとファイルの区切り記号としてカンマや空白を使用していることや、ワイルドカードの記号とプログラムが判断してしまうことがあるかもしれないからである。

1.5. ステージング仕様

ステージングの仕様は cp コマンドのオプションなしと同等な動作である。ただし、ハッシュを使用したファイルステージングと転送先を指定しなかった場合のファイルステージングの挙動は cp コマンドとは違う挙動になる。

- 転送元にはファイル名を指定する。
- 転送元にディレクトリを指定することはできない。
- 転送先にはファイル名、ディレクトリ名を指定できる。
- 転送元にワイルドカードを指定できる。
- 使用できるワイルドカードは“＊”と“？”である。
- ディレクトリ名を指定時はディレクトリ名の後ろに”/”を付ける

- ディレクトリは相対パス、完全パスでの指定ができる。
- ディレクトリ名を含めたファイル名の指定ができる。
- 転送したファイルの所有者はファイル転送を行ったユーザになる。
- 転送先に何も指定しなかった場合、転送先の基準ディレクトリに転送元ファイルを転送する。

1.6. 指定方法

ファイルステージングの指定方法にはいくつかの指定方法があり、ユーザの用途に合わせて指定方法を選択することができる。以下の図はファイルステージング指定方法を簡略化した図である。

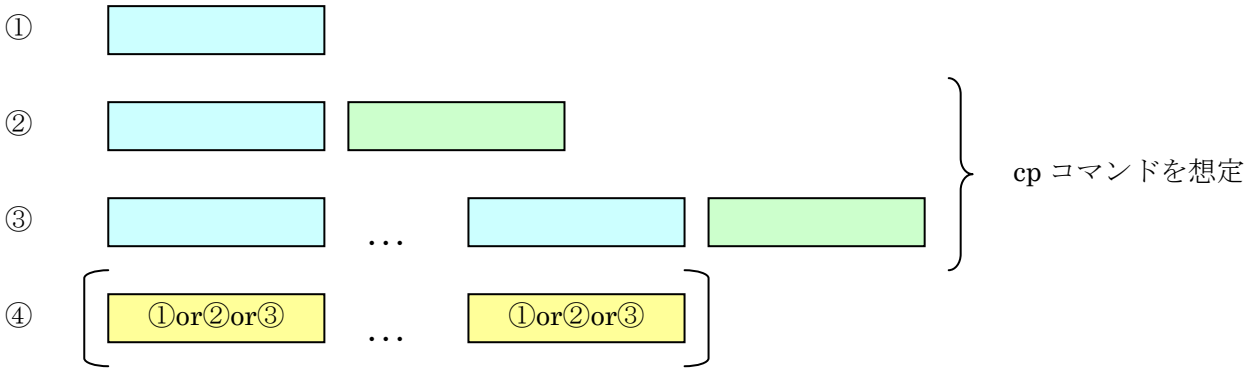

図 3　ファイルステージング指定方法簡略図

次に図の説明を記述する。

- 水色のブロック

水色のブロックには転送元ファイル名を記述する。転送元は「相対パス指定」、「完全パス指定」、「ワイルドカード指定」が記述可能である。転送元にディレクトリを指定することはできない。

- 緑色のブロック

緑色のブロックには転送先ファイル名、もしくは転送先ディレクトリ名を記述する。転送先は「相対パス指定」、「完全パス指定」、「ファイル名変更」が記述可能である。転送先にワイルドカードを指定することはできない。

- ①

水色ブロックを１つ指定したファイルステージング方法。単数ファイルを転送する。パスは転送元と転送先で同じ階層になる。記述方法はスカラ文字列指定、サブルーチン指定が可能。

例

ログインノードのカレントディレクトリから計算ノードのカレントディレクトリへ「foo.txt」をスカラ文字列指定でステージインする。

```
JS_stage_in_files => “./foo.txt”
```

- ②

水色ブロック１つと緑色ブロック１つを指定したファイルステージング方法。転送元と転送先がはっきりと記述することができる。cp コマンドと同等な動作になる。記述方法はスカラ文字列指定、ハッシュ指定、サブルーチン指定が可能。

例

ログインノードの[work/]内の「var.txt」を計算ノードのカレントディレクトリに[hoge.txt]というファイル名でステージインする。

```
JS_stage_in_files => [ “work/var.txt”, “./hoge.txt” ]
```

- ③

水色ブロックを複数と緑色ブロックを１つ指定したファイルステージング方法。転送元の複数個のファイルを転送先のディレクトリに転送する。この場合、緑色のブロックには「ディレクトリ名」のみ記述が可能になる。cp コマンドと同等な動作になる。記述方法はスカラ文字列指定、サブルーチン指定が可能。

例

ログインノードのカレントディレクトリ内の「fuga」で始まるファイル(ワイルドカード)と「tmp/」内の「piyo.txt」を計算ノードの「data/」にステージインする。

```
JS_stage_in_files => [ “./fuga*”, “tmp/piyo.txt”, “data/” ]
```

- 黄色ブロック

①、②、③いずれかの指定がされたブロック。

- ④

黄色ブロックを複数個指定したファイルステージング方法。転送の組み合わせを 1 回で記述することが可能。

例

①、②、③の例を１度に記述する。

```
JS_stage_in_files => [ “./foo.txt”, [ “work/var.txt”, “./hoge.txt” ], [ “./fuga*”, “tmp/piyo.txt”, “data/” ]]
```

1.6.1. ハッシュ指定

‘[]’でファイルを囲いアレイのリストとしてその中にハッシュキーを使用してステージングファイルを指定する。ハッシュキーとして‘local_file’と‘remote_file’がある。ローカル側のパス指定（ステージインの時の転送元とステージアウトの時の転送先）は‘local_file’に記述する。リモート側のパス指定（ステージインの時の転送先とステージアウトの時の転送元）は‘remote_file’に記述する。以下が記述例である。ハッシュ指定の時’local_file’、’remote_file’の両方を使用し、どちらかだけ指定しないといったことはできない。

```
‘JS_stage_in_files’ => [{‘local_file’ => ‘staging_file1’, ‘remote_file’ => ‘work/staging_file2’}]
```

1.6.2. スカラー文字列指定

シングルクオートかダブルクオートでステージングファイルを囲い文字列として’JS_stage_in_files’または、’JS_stage_out_files’に指定する。カンマ区切りでファイル(ディレクトリ)を複数指定する。以下が記述例である。

```
‘JS_stage_in_files’ => ‘staging_file, work/’
```

1.6.3. サブルーチン指定

ジョブの数を使った指定ができる。ジョブの数はジョブ定義ハッシュの‘RANGE’で指定されている。指定値として‘$VALUE’を用いる。

以下のようなXcryptスクリプトを例に説明する。

```
use base qw(core);

%template = (
    'id' => 'job0',
    'RANGE0' => [0..9],
    'JS_stage_in_files@' => sub{"staging_file$VALUE[0]"},
    'exe0@' => sub{ “a.out staging_file$VALUE[0]” },
);

my @jobs = &prepare_submit_sync(%template);
```

ジョブ定義ハッシュ‘RANGE’には0~9の値が設定されている。つまりジョブはjob0_0~job0_9まで10個が作成される。そのジョブ毎のRANGEの変化値が$VALUE[0]には入っている。job0_0の時、ステージングファイルは'staging_file0'になり、ジョブ投入ディレクトカレントの‘staging_file0’をベースディイレクトリに転送する。ジョブ毎に値が違うファイルを転送する。

以下が記述例である。RANGE0 => [0..1]、RANGE1 => [2..3]とする。

@なし、@あり両方の記述が可能である。

```
‘JS_stage_in_files’ => sub{ “foo$VALUE[0].txt” }
‘JS_stage_in_files@’ => sub{ “foo$VALUE[0].txt” }
```

第2章 ファイルステージングユーザ記述例

ユーザスクリプトへの記述の仕方を以下に示す。

- ログインノードのカレントから計算ノードのカレントへ「foo.txt」をステージインする。

```
JS_stage_in_files => “foo.txt”
```

- ログインノードのカレントから計算ノードのカレントへ「foo.txt」を名前を変更してステージインする。

```
JS_stage_in_files => [“foo.txt”, “hoge.txt”]
```

上記の方法と同じ転送で〔 〕をはずし、全体をダブルクオートで括る記述も可能。

```
JS_stage_in_fils => “foo.txt, hoge.txt”
```

- [“foo.txt”, “foo.txt”]のシンタックスシュガー。

```
JS_stage_in_files => [“foo.txt”]
```

- ログインノードのカレントから計算ノードのカレントへ複数のファイルをステージインする。「foo.txt」「hoge.txt」「fuga.txt」を「./」に転送する。

```
JS_stage_in_files => [“foo.txt”, “hoge.txt”, “fuga.txt”, “./”]
```

- ログインノードの「./work/」から計算ノードのカレントへ「foo.txt」をステージインする。

```
JS_stage_in_files => “./work/foo.txt, ./foo.txt”
```

上記の方法と同じ転送でハッシュ指定を使用し転送元と転送先をはっきりと表す事も可能。

```
JS_stage_in_files => [{local_file => “./work/foo.txt”. remote_file => “./foo.txt”}]
```

- ログインノードのカレントから計算ノードの「./data/」に「foo.txt」を転送する。

```
JS_stage_in_files => [“foo.txt”, “./data/”]
```

- ログインノードの「/tmp」内の「foo.txt」を計算ノードのカレントにステージインする。

```
JS_stage_in_files => [“/tmp/foo.txt”, “./”]
```

- ワイルドカードを使用し複数ファイルをステージインする。

ログインノードのカレント内、「foo*.txt」に該当するファイルを計算ノードの「work/」にステージインする。

```
JS_stage_in_files => [“foo*.txt”, “work/”]
```

上記と同じステージングを[ ]を取りダブルクオートで括る記述も可能。

```
JS_stage_in_files => “foo*.txt, work/”
```

上記と同じステージングをハッシュ指定を使用して記述も可能。

```
JS_stage_in_files => [{local_file => “foo*.txt”, remote_file => work/}]
```

- ログインノードの「work/」内、「foo?.txt」に該当するファイルを計算ノードのカレントにステージインする。

```
JS_stage_in_files => [“work/foo?.txt”, “./”]
```

● サブルーチン指定(RANGE の変化値)を使用してステージインする。

ログインノードのカレントにある[foo0.txt]から「foo5.txt」のファイルを計算ノードの「work/」にステージインする。

```
RANGE0 => [0..5],
JS_stage_in_files => [sub{ “foo$VALUE[0].txt” }, “work１/” ]
```

上記と同じ転送で@マークをキーの後ろに書き、ステージング対象全体を sub で囲う記述方法もある。

```
RANGE0 => [0..5],
JS_stage_in_files@ => sub{[ “foo$VALUE[0].txt”, “work/” ]}
```

● サブルーチン指定してファイル名を変更してステージインする。

ログインノードのカレントにある「foo0.txt」から「foo2.txt」のファイルを計算ノードのカレントに「hoge0.txt」から「hoge2.txt」のファイル名に変更してステージインする。

```
RANGE0 => [0..2],
JS_stage_in_files => [sub{ “foo$VALUE[0].txt” }, sub{ “hoge$VALUE[0].txt” }]
```

● サブルーチン指定と相対パスを組み合わせてステージインする。

ログインノードの「../up/」にある「foo0.txt」から「foo3.txt」のファイルを計算ノードのカレントへステージインする。

```
RANGE0 => [0..3],
JS_stage_in_files => [sub{ “../up/foo$VALUE[0].txt” }, “./” ]
```

● RANGE の変化値を２つ使用する。

「RANGE0=>[0..1]」と「RANGE1=>[2..3]」があり、「foo02」、「foo03」、「foo12」、「foo13」という４つのファイルをジョブ毎に計算ノードの「work/」にステージインする。

```
RANGE0 => [0..1],
RANGE1 => [2..3],
JS_stage_in_files => [sub{ “foo$VALUE[0]$VALUE[1]” }, “work/” ]
```

● 複数のファイルステージングを記述する。

ログインノードのカレントにある「aaa」「bbb」「ccc」を計算ノードの「abc/」にステージインする。ログインノードの「work/」にある全ファイルを(ワイルドカード)を計算ノードのカレントにステージインする。

```
JS_stage_in_files => [[ “aaa”, “bbb”, “ccc”, “abc/” ], [ “work/*”, “./” ]]
```

● data_generator と data_extractor との併用。

data_generator で使用する雛形ファイル(HPL.dat)をステージインする。after_in_job で data_extractor を使用する。data_extractor した結果を格納した「job1_return」をステージアウトする。ステージアウトした「job1_return」を after で参照する。

```
use base qw(file_stager core);
use data_generator;
use data_extractor;
%tmplate(
    id => “job1”,
    ’exe0’ => “./a.out”
    JS_stage_in_files => [ “HPL.dat ” ],
    JS_stage_out_files => [ “job1_return” ],
```

```
    before_in_job => sub{
        my $self = shift;
        system("mkdir $self->{id}");
        my $hpldat = data_generator -> new("HPL.dat", $self->{id} . "/HPL.dat");
        $hpldat -> replace_line_column ( 9, 1, 1);
        $hpldat -> replace_line_column (11, 1, 2);
        $hpldat -> replace_line_column (12, 1, 3);
        $hpldat -> execute();
    },
    after_in_job => sub{
        my $self = shift;
        my $stdout_file = "/$self->{id}_stdout";
        my $stdout = data_extractor->new($stdout_file);
        $stdout -> extract_line_rn (['PASSED',-2]);
        $stdout -> extract_line_nn (1);
        $stdout -> extract_column_nn (7);
        @result = $stdout -> execute();
        return ¥@result;
    },
    after => sub{
        my $self = shift;
        my @result = $self->get_after_in_job_return();
        print "@result¥n";
    },
);
&prepare_submit_sync(%template);
```