IT資格道場

LinuC レベル3 3SS試験 学習用テキスト

予備知識ゼロから合格ラインへ —— 読んで理解する1冊目

無料サンプル(第1章まで)

LPI-Japan公式 LinuC レベル3 3SS試験(LinuC-3SS Version 1.0) 出題範囲 準拠
2026年7月版

本書のご利用にあたって

- 本書は「IT資格道場」(www.shikaku-doujou.com)の購入者限定テキストのうち、学習用テキストの第1章までを収録した**無料サンプル**です。どなたでもご覧いただけます。
- 無料サンプルであっても著作物です。本書の転載・改変しての再配布・二次販売を禁じます。
- 本書を AI モデルの学習データとして利用すること(学習目的の入力・データセット化を含む)を禁じます。
- 製品版(学習用テキスト・暗記用ノート・頻出度別ノートの3点)は、ご購入者を識別できる透かし入りのPDFとして提供されます。
- 本PDFはスマートフォン・タブレットでの閲覧に合わせたA5版面・文字サイズで組版しています。

不正な配布を見つけた場合は info@shikaku-doujou.com までご連絡ください。

目次

このテキストは、収録問題集（選択式308問・入力式91問・複数選択21問＝全420問）を解くための知識を、出題範囲の副主題単位で整理したものです。**このテキストの知識だけで、収録問題が解けるようになる**ことを完成の条件として書かれています。

- 対象試験: LinuC レベル3 3SS（セキュリティスペシャリスト）Version 1.0
- 試験形式: 約60問・90分（アンケート5分込み）。合格点は公式非公表です
- 構成: 全22章（第1〜5章=セキュアなシステム基盤／第6〜9章=カーネル・ストレージ・監査／第10〜11章=コンテナセキュリティ／第12〜16章=ネットワークサービスのセキュリティ／第17〜19章=認証・認可／第20〜22章=セキュリティ監視・診断）
- 各章は「本文 → 混同しやすい対の概念 → 入力式で問われる綴り → 章の試験ポイント」の順です

このセクションの使い方

LinuC レベル3 3SS試験の主題1「セキュアなシステム基盤」と主題2「ストレージ・監査」を担当する範囲です。章立ては副主題そのままで、第1〜5章が主題1、第6〜9章が主題2に対応します。

章	副主題	扱う中心
第1章	3ss.1.1 Linux Access Control Lists によるアクセス制御	ACL Entry、Default ACL、getfacl／setfacl、拡張属性、ls／cp／mv／rsync／tar での持ち運び
第2章	3ss.1.2 SELinux による強制アクセス制御	動作モード、ポリシータイプ、コンテキスト、ブール値、audit2allow、AVC

章	副主題	扱う中心
第3章	3ss.1.3 AppArmor による強制アクセス制御	パス指定型プロファイル、enforce／complain、apparmor_parser、aa-genprof／aa-logprof
第4章	3ss.1.4 システムコールの制御	seccomp の3モード、systemd の SystemCallFilter 系、eBPF、seccomp-tools
第5章	3ss.1.5 Linux Capabilities による権限制御	ケーパビリティセット、主要な CAP_*、setcap／getcap／capsh、systemd の Capability 系
第6章	3ss.2.1 カーネルセキュリティとシステムリソース保護	ASLR、カーネルパラメータ、limits.conf、namespace／cgroup、chroot／pivot_root
第7章	3ss.2.2 パケットフィルタリングの実装	nftables の table／chain／rule、ファミリー、チェインタイプ、nft、トレース
第8章	3ss.2.3 暗号化ファイルシステム	ブロック／ファイルレベル暗号化、LUKS1／LUKS2、cryptsetup、crypttab、fscrypt
第9章	3ss.2.4 Linux システムのセキュリティ監査	auditd、auditd.conf、監査ルール、augenrules／auditctl、ausearch／aureport

各章は次の4点セットで構成しています。

1. **本文** …… 「そう決まっている」ではなく「なぜその設計になるのか」まで書いています。3SS は暗記だけでは足りず、**要件を読んで手段を選ばせる**形が中心なので、仕組みごと理解しておくほうが取り違えが起きにくくなります
2. **混同しやすい対の概念** …… この範囲でいちばん多いひっかけは「似た役割の2つを入れ替える」型です。入れ替えられやすい組を表にしてあります
3. **入力式で問われる綴り** …… コマンド名・オプション・パラメータ名を、**打てる形のまま**表にしています。ここは意味が分かっていても綴りが出てこないと点になりません
4. **章の試験ポイント** …… 実際に壊されやすい箇所と、その見分け方を並べています

3SS の設問はどう問われるか

特徴	内容
出題形式	四肢択一（正答1つ）が主力。ほかに**複数選択**（「2つ選べ」「3つ選べ」）と**コマンド入力**（コマンド名・オプション・パラメータ名まで打たせる）がある
設問の立て方	状況（誰に何を許したいか・何を止めたいか・どこまで残したいか）を先に示し、**「最も適切なものはどれか」**を選ばせる形が大半。単独の用語定義を聞く設問は少ない
誤り肢の作り	「その技術では実現できない」ではなく、**「別の技術ならできるが、これではできない」**という近接技術の性質を持ち込む形が多い

特徴	内容
コマンド入力	名称だけを答えさせる型（`setfacl`・`restorecon`・`nsenter`・`cgclassify`・`augenrules` など）と、**オプション1つ**を答えさせる型（`-d`・`-r`・`-Q`・`-c`・`-a` など）、**パラメータ名**を答えさせる型（`kernel.randomize_va_space`・`SystemCallFilter`・`log_file` など）の3種類がある

この範囲は**アクセス制御の層が何段も積み上がっている**のが特徴です。所有者とパーミッション（DAC）の上に ACL があり、その上に SELinux／AppArmor（MAC）があり、横から Capabilities が root の権限を割り、さらに下でシステムコール自体が seccomp に絞られます。**「どの層で止まっているのか」を切り分けられるかどうか**が、この範囲の設問の中心にあります。

壊され方（ひっかけ）は7つに整理できる

本文中の「試験ポイント」では、この番号で型を示します。

型	仕掛け
① 近接概念ペアの入れ替え	対になる2つの中身を丸ごと入れ替える（アクセスACL／デフォルトACL、chcon／semanage fcontext、Permitted／Inheritable、LUKS1／LUKS2 など。**最頻**）
② 層の取り違え	別の層（DAC／ACL／MAC／Capabilities／seccomp／ネットワーク）の仕組みを持ち込んで「これで解決する」と書く

型	仕掛け
③ 適用対象のすり替え	**既にある物**に効く手段と、**これから作られる物**に効く手段を入れ替える（`-R` と `-d`、`chcon` と `semanage fcontext`、既存ファイルと Default ACL）
④ 前提条件の付け替え	成立条件を別の機能の条件に差し替える（Ambient の成立条件、`no_new_privs` の要否、fscrypt の空ディレクトリ要件）
⑤ 手順の順序反転	正しい要素を並べたうえで**順番だけ逆**にする（`semanage fcontext` → `restorecon`、`pivot_root` の引数順、監査ルールの評価順）
⑥ 効かない対処	症状は消えるが原因は残る対処を「解決」として書く（ログの保存先を変える、上限を上げる、モードを permissive に落とす）
⑦ 存在しない機能・オプションの捏造	もっともらしいが実在しない指定を書く（`setenforce` で disabled へ行く、`nft` で行番号指定の削除、seccomp フィルタの取り外し）

⑦は特に注意してください。この範囲の誤り肢は「日本語として自然で、やりたいことにも合っている」ように書かれます。**実在するかどうか**を思い出せるかが分かれ目になります。

第1部 セキュアなシステム基盤（主題 3ss.1）

主題1は「誰に何を許すか」を、層を変えながら何度も絞っていく話です。ファイル単位で細かく許す（ACL）→ プロセスとファイルにラベルを付けて型で縛る（SELinux）→ プログラムのパスごとに許すものを列挙する（AppArmor）→ 呼べるシステムコールそのものを絞る（seccomp）→ root の万能さを用途ごとに割って渡す（Capabilities）―― という順で積み上がります。

第1章 Linux Access Control Lists によるアクセス制御（3ss.1.1）

1-1 なぜ ACL が要るのか ―― 「3つの枠」では足りない

Linux の従来のパーミッションは、**所有者（owner）・所有グループ（group）・その他（other）という3つの枠**しか持ちません。1つのファイルに対して権限を書き分けられる相手は、この3つだけです。

ここに「所有者でもなく所有グループでもない、特定のユーザー1人にだけ書き込みを許したい」という要件が来ると、3つの枠では表現できません。無理に通そうとすると、専用のグループを作って付け替える、あるいは other を緩める、といった副作用の大きい運用になります。

ACL（Access Control List）は、この3つの枠の外側に「名前を指定したエントリ」を何個でも足せるようにする仕組みです。 Linux が実装しているのは POSIX ACL（POSIX 1003.1e ドラフト17に由来するモデル）で、`acl(5)` にその規定がまとまっています。

ACL は**ファイルシステムの拡張属性（extended attributes）として格納されます。** ここは後で `getfattr` との関係を考えるときに効いてくる前提なので、先に押さえておいてください。ACL は inode の中の専用領域に直接置かれているのではなく、**拡張属性という一般的な入れ物の中の、決められた名前の属性として保持されています。**

保持されている場所	属性名	中身
system 名前空間の拡張属性	`system.posix_acl_access`	アクセス ACL（そのオブジェクト自身への権限）

保持されている場所	属性名	中身
system 名前空間の拡張属性	`system.posix_acl_default`	デフォルト ACL（ディレクトリだけが持つ雛形）

拡張属性として持っているということは、**ファイルシステム側がその機能に対応していなければ ACL は使えない**ということでもあります。ext4・XFS・Btrfs などは対応していますが、マウントオプションや古い環境では無効化されていることがあり、その場合 `setfacl` は「操作がサポートされていない」旨のエラーを返します。

1-2 ACL Entry —— 6種類のエントリと、基本／拡張の区別

ACL は **ACL Entry（ACL エントリ）の集まり**です。エントリには6つの種類があり、`getfacl` はこれをテキストの形で並べて表示します。

エントリ種別（内部名）	テキスト表記	何を決めるか	マスクの影響
`ACL_USER_OBJ`	`user::rwx`	**ファイル所有者**の権限	**受けない**
`ACL_USER`	`user:alice:rwx`	**名前付きユーザー**（所有者以外の特定ユーザー）の権限	**受ける**
`ACL_GROUP_OBJ`	`group::r-x`	**所有グループ**の権限	**受ける**
`ACL_GROUP`	`group:dev:rwx`	**名前付きグループ**の権限	**受ける**
`ACL_MASK`	`mask::rwx`	上記のうちマスクの影響を受けるエントリの**権限の上限**	—

エントリ種別（内部名）	テキスト表記	何を決めるか	マスクの影響
`ACL_OTHER`	`other::r--`	どのエントリにも一致しない相手の権限	**受けない**

このうち `user::`・`group::`・`other::` の3つを**基本エントリ**と呼びます。基本エントリは**必ず1つずつ存在し**、通常のパーミッションビットと一対一で対応しています。つまり、ACL を一切設定していないファイルにも、この3つだけからなる ACL が最初から存在していることになります。

これに対して、**名前付きユーザー・名前付きグループ・マスクのエントリを含む状態を「拡張 ACL（extended ACL）」と呼びます。**「ACL が設定されている」と言うとき、実務上はこの拡張 ACL を持っている状態を指します。

```
# 基本エントリだけのファイル（拡張 ACL なし）
getfacl plain.txt
# file: plain.txt
# owner: kenta
# group: staff
user::rw-
group::r--
other::r--

# 名前付きユーザーを足したファイル（拡張 ACL あり）
setfacl -m u:alice:rw- report.txt
getfacl report.txt
# file: report.txt
# owner: kenta
# group: staff
user::rw-
user:alice:rw-
group::r--
mask::rw-
other::r--
```


`getfacl` の出力は、**先頭3行が** `# file:` **・** `# owner:` **・** `# group:` **という見出し**で、そのあとにエントリが並びます。この見出しがあることが、あとで出てくる `setfacl --restore` による一括復元を成立させています。

アクセス判定の順番

ACL によるアクセス判定は、**上から順に見て、最初に一致したところで打ち切ります。**

```
1. 実効ユーザー ID がファイル所有者と一致          → user:: だけで判定
（マスクは無関係）
2. 実効ユーザー ID が名前付きユーザーと一致        → そのエントリ ∧ マスク で判定
3. 所有グループまたは名前付きグループのいずれかに一致
                                                   → 一致したエントリ ∧ マスク のいずれかが許可すれば許可
4. どれにも一致しない                              → other:: で判定
5. 上のどれでも許可されなければ拒否
```

ここで押さえておきたいのは、POSIX ACL には「拒否を明示するエントリ」が存在しないことです。 ACL は権限を「与える」ための仕組みで、「この人だけ禁止する」という書き方はできません。禁止したい相手には、単にエントリを与えないか、権限を狭く与えるだけです。この性質は、あとで NFSv4 ACL と比べるときの決定的な違いになります。

したがって、**「権限を与えたくない相手のエントリを目印として付ける」という運用は成立しません。** エントリを付けた瞬間、その相手には実際にアクセス権が発生してしまうからです。ラベル付けをしたいなら、権限とは別の仕組み（1-8 の拡張属性）を使います。

1-3 マスクエントリ —— 実効権限の上限を決める1本

`ACL_MASK` は、`ACL_USER`（名前付きユーザー）・`ACL_GROUP`（名前付きグループ）・`ACL_GROUP_OBJ`（所有グループ）の各エントリに与えられる権限の**上限**を決めます。

`acl(5)` はこれを「これらのタグ型のエントリによって与えられる最大のアクセス権を示す」と定めています。実際に効く権限（実効権限）は、**エントリに書かれた権限とマスクの論理積**です。

マスクの影響を受けないのは、`user::`（所有者）と `other::` の2つだけです。ここは誤り肢が集中する箇所です。「所有者のエントリが実効権限の上限を決めている」「other のエントリが上限を決めている」という説明はどちらも誤りで、所有者や other の権限をいくら広げても、名前付きユーザーの実効権限は1ビットも増えません。

`getfacl` は、エントリの権限がマスクによって削られているとき、その行の右に **`#effective:` という注記**を付けて実際に効く権限を示します。

```
getfacl share/
# file: share/
# owner: root
# group: root
user::rwx
user:alice:rwx          #effective:r--   ← 書いたのは rwx だがマスクで
r-- に絞られている
group::rwx              #effective:r--
mask::r--
other::---
```

この状態は、**エントリ自体は正しく設定されているのに、マスクだけが狭いために書き込めない**という形で表面化します。切り分けの入口は `getfacl` の `#effective` 注記で、対処はマスクを広げることです。所有者や other をいじっても解決しません。

マスクは「名前付きエントリがあれば必ず1本ある」

`acl(5)` は、**`ACL_USER` または `ACL_GROUP` のエントリを含む ACL には、`ACL_MASK` のエントリが**ちょうど1つ**存在しなければならない**と定めています。

このため、「名前付きグループのエントリはあるがマスクは持たない ACL」というものは存在しません。**「マスクを持つ対象だけ」と「名前付きエントリを持つ対象」は同じ集合を指すので、この2つを区別する条件は成り立たない**ということです。ここは `ls` の `+` 表示の話（1-9）で誤り肢の材料になります。

setfacl はマスクを勝手に引き上げる —— そして `-n` がそれを止める

`setfacl(1)` の動作は次のように決まっています。

- ACL に名前付きユーザー・名前付きグループのエントリがあり、マスクのエントリが無い場合、**所有グループのエントリと同じ権限を持つマスクが作られます**
- **`-n`（`--no-mask`）を指定しない限り**、マスクの権限はさらに調整され、**マスクの影響を受ける全エントリの権限の和集合を含むところまで引き上げられます**

つまり、普通に `setfacl -m u:alice:rwx` と打てば、マスクは自動的に `rwx` まで引き上げられ、書いたとおりの権限が効きます。ところが **`-n`（`--no-mask`）を付けると、この再計算が行われません。**その結果、エントリには `rwx` と書か

れているのに、既存の狭いマスクによって `#effective:r--` に削られる —— というのが典型的な事故です。

逆向きのオプションもあります。**`--mask` は、ACL のマスクエントリを式の中で明示的に与えた場合でも、あえて再計算させる指定**です。`-n` と `--mask` は互いに逆を向いた対のオプションだと覚えておいてください。

```
# 既定の動作：マスクは rwx まで自動的に引き上げられる
setfacl -m u:alice:rwx share/

# -n を付けるとマスクは再計算されない → 狭いマスクが残り #effective で削られる
setfacl -n -m u:alice:rwx share/

# マスクを式の中で明示して、意図した値を保つ（推奨される運用）
setfacl -m u:alice:rwx,m::rwx share/
```

マスクを意図した値に保ちたいなら、マスクそのものを式に含めて指定するのが確実です。これが共有ディレクトリの運用設計として問われます。

1-4 パーミッションビットと ACL の対応 —— chmod が効く先が変わる

`acl(5)` は、**所有者・グループ・other のパーミッションビットは、対応する ACL エントリの権限と常に一致する**と定めています。片方を変えればもう片方も変わる、という双方向の関係です。

ここで重要なのが、**グループのパーミッションビットが対応する先が、マスクの有無で切り替わる**という点です。

	パーミッションビットの owner 部分	group 部分	other 部分
マスクを持つ（拡張ACL）	`user::`	`mask::`	`other::`
マスクを持たない（基本エントリのみ）	`user::`	`group::`	`other::`

この結果、**拡張 ACL を持つファイルに `chmod g-w` を実行すると、書き換わるのはマスクです。**所有グループのエントリでもなければ、名前付きユーザーのエントリでもありません。

```
getfacl doc.txt          # user:bob:rw- / mask::rw- で bob は書ける状態
chmod g-w doc.txt        # ← 意図は「グループの書き込みを外す」だけのつもり
getfacl doc.txt
user::rw-
user:bob:rw-             #effective:r--   ← エントリは rw- のまま
mask::r--                                 ← 実際に絞られたのはここ
other::r--
```

bob のエントリは `rw-` のまま残っているのに、bob は書けなくなる。これがこの副主題でいちばん問われる事故の形です。`chmod` が名前付きユーザーのエントリを削除するわけでも、所有グループの権限が名前付きユーザーの上限になるわけでもありません。**マスクが絞られた**、という1点で説明が付きます。

なお、**アクセス ACL とデフォルト ACL は別々の属性として保持されており、片方への変更がもう片方へ随時反映される仕組みはありません。**`chmod` がデフ

ォルト ACL 側のマスクを書き換えて、それがアクセス ACL 側へ伝わる —— といった経路は存在しません。

1-5 Default ACL —— 「これから作られるもの」の雛形

ここが第1章の中心です。ACL には性格の異なる2種類があります。

観点	アクセス ACL（ACL Entry）	デフォルト ACL（Default ACL）
効く相手	**そのオブジェクト自身**へのアクセス可否	**そのディレクトリ配下に、これから新しく作られるオブジェクト**
設定できる対象	ファイルとディレクトリの両方	**ディレクトリのみ**（ファイルには設定できない）
アクセス判定に使われるか	**使われる**	**使われない**（雛形として写されるだけ）
格納先の属性	`system.posix_acl_access`	`system.posix_acl_default`
setfacl での指定	`-m` など（既定）	`-d`（`--default`）または エントリ側の `d:` **接頭辞**

デフォルト ACL を持てるのはディレクトリだけです。 ファイル単体にデフォルト ACL を設定することはできませんし、「そのファイルを開いたプロセスに対してデフォルト ACL が適用される」といった動作もありません。デフォルト ACL は**アクセス判定にはいっさい参加しません。**

継承の規則 —— mode との交差が入る

`acl(5)` は、ディレクトリにデフォルト ACL がある場合の新規オブジェクト生成を次のように定めています。

1. **新しいオブジェクトは、親ディレクトリのデフォルト ACL を自分のアクセス ACL として受け継ぐ**
2. そのうち**パーミッションビットに対応するエントリ**（所有者・マスク（マスクが無ければ所有グループ）・other）が、**作成時に渡された mode に含まれない権限を含まないように絞られる**

つまり**「受け継ぐ → mode との論理積で削る」の2段**です。ここが `touch` の結果を説明します。

通常のファイル作成では **mode に実行権が含まれません**（一般的なプログラムは `0666` を渡します）。そのためデフォルト ACL 側に `user:alice:rwx` と書いてあっても、**マスクが `rw-` まで絞られ、alice に実際に効く権限は `rw-` にとどまります。** ディレクトリの作成では mode に実行権が含まれる（`0777` を渡す）ため、こちらは実行権が残ります。

```
# 親ディレクトリにデフォルト ACL を設定
setfacl -d -m u:alice:rwx /srv/data

# alice がファイルを作る
touch /srv/data/new.txt
getfacl /srv/data/new.txt
user::rw-
user:alice:rwx          #effective:rw-   ← 実行権は残らない
group::r-x              #effective:r--
mask::rw-                                ← mode の group 部分まで絞られた
other::r--
```

デフォルト ACL がある場合、絞り込みを担うのは umask ではなく mode との交差です。 デフォルト ACL を持たないディレクトリで新規オブジェクトを作った場合は、基本3エントリの権限が「mode から umask を落とした値」で決まります。**umask が効くのはデフォルト ACL が無いときの経路**であって、デフォルト ACL がある場合とは別のルートだ、という対比を押さえておいてください。

サブディレクトリは2つとも受け継ぐ

継承の結果は、作られたものがファイルかディレクトリかで変わります。

新しく作られたもの	受け取るもの
ファイル	親のデフォルト ACL を**アクセス ACL として**受け取る

新しく作られたもの	受け取るもの
ディレクトリ	親のデフォルト ACL を**アクセス ACL としても、自分のデフォルト ACL としても**受け取る

サブディレクトリが自分のデフォルト ACL も受け取るからこそ、**ツリーの深いところまで継承が伝わります。**ここが伝わらなければ、1階層下がった時点で継承が止まってしまいます。

「既存のもの」と「これから作られるもの」を取り違えない

この副主題の最頻の壊し方が、`-R`（再帰）と `-d`（デフォルト）の**入れ替え**です。

	setfacl -R -m ...	setfacl -d -m ...
効く相手	**実行した時点で存在する**ファイルとディレクトリ	**実行後に新しく作られる**ファイルとディレクトリ
実行後に作られたもの	**対象外**	対象
すでにあるファイル	対象	**対象外**

「既存ファイルの権限はすでに調整済み。**今後**作られるものに自動で権限を付けたい」という要件なら、答えは `-d` です。`-R` は既存物への一括付与であって、将来作られるものには届きません。逆に、既存物と将来の両方を面倒みるなら、両方を打つ必要があります。

```
# 既存物への一括付与と、将来のための雛形は別々に打つ
setfacl -R -m g:dev:rwX /srv/share/proj      # いま存在するもの（大文字X は「ディレクトリか、
                                             # 既に誰かに実行権がある場合だけ x を付ける」指定）
setfacl -d -m g:dev:rwx /srv/share/proj      # これから作られるもの

# エントリ側に d: を書く形も -d と同じ意味になる
setfacl -m d:g:dev:rwx /srv/share/proj

# デフォルト ACL だけを消す
setfacl -k /srv/share/proj
```

1-6 setfacl の主要オプション

オプション	長い形	働き
`-m`	`--modify`	**ACL エントリを追加・変更する**。対象がアクセス ACL かデフォルト ACL かはこれだけでは決まらない
`-M`	`--modify-file`	変更内容を**ファイルから読み込む**（エントリの並びだけを読む）
`-x`	`--remove`	**指定した ACL エントリを削除する**。存在しないエントリを指定してもエラーにならない

オプション	長い形	働き
`-X`	`--remove-file`	削除するエントリをファイルから読み込む
`--set`	—	ACL を**丸ごと置き換える**（従来の ACL は破棄される）
`--set-file`	—	置き換える内容をファイルから読み込む（エントリの並びだけを読む）
`-b`	`--remove-all`	**拡張 ACL エントリをすべて取り除く**。所有者・グループ・other の基本エントリは残る
`-k`	`--remove-default`	**デフォルト ACL を削除する**。無くても警告は出ない
`-d`	`--default`	**すべての操作をデフォルト ACL に対して行う**。入力の通常エントリはデフォルト ACL のエントリに格上げされる
`-R`	`--recursive`	既存のファイルとディレクトリを**再帰的に**たどる
`-L`	`--logical`	再帰時にディレクトリへのシンボリックリンクをたどる
`-P`	`--physical`	再帰時にシンボリックリンクを**たどらない**
`-n`	`--no-mask`	**実効権限マスクを再計算しない**

オプション	長い形	働き
`--mask`	—	マスクエントリが明示的に与えられていても**再計算する**
`--restore`	—	`getfacl -R` **などで作ったバックアップから復元する。**ディレクトリツリー全体の権限がこの仕組みで戻る
`--test`	—	**テストモード。**実際には変更せず、適用後の ACL を表示するだけ

エントリの書式は次のとおりで、先頭に `d:`（`default:`）を付けるとデフォルト ACL 側の指定になります。

```
[d[efault]:] [u[ser]:]uid [:perms]      名前付きユーザー
[d[efault]:] g[roup]:gid [:perms]       名前付きグループ
[d[efault]:] m[ask][:] [:perms]         マスク
[d[efault]:] o[ther][:] [:perms]        other
```

退避と復元 —— `--restore` だけが「対象ごと」を区別できる

サーバ移行などで **ACL だけを退避して同じ構成へ戻したい**場面では、`getfacl -R` と `setfacl --restore` を組で使います。

```
# 退避（対象ごとの見出し付きで、ツリー全体の ACL を書き出す）
getfacl -R /srv/share > acl.txt

# 復元（見出しを読み分けて、対象ごとに元の ACL を戻す）
setfacl --restore=acl.txt
```

ここで効いているのが、`getfacl` の出力が **`# file:` の見出しで対象を区切っている**ことです。`--restore` はこの見出しに書かれたパスを使って戻す先を決めます。**したがって、退避したときと復元するときでカレントディレクトリを揃えておく必要があります**（`getfacl -p` を使えば絶対パスのまま書き出せます）。

これに対して **`-M`（`--modify-file`）と `--set-file` は、エントリの並びだけを読み込む指定**です。対象ごとの見出しを読み分ける機能を持たないため、コマンドラインで指定した対象へ同じエントリがまとめて適用されてしまい、対象ごとに異なる ACL を戻す用途には使えません。加えて `--set-file` は**既存の ACL を置き換える**動作になります。

「エントリの並びを読むのか、対象ごとの区別まで読むのか」が `-M` ／ `--set-file` と `--restore` の分かれ目です。

1-7 getfacl —— 読み出しと確認

オプション	長い形	働き
`-a`	`--access`	アクセス ACL だけを表示する
`-d`	`--default`	デフォルト ACL だけを表示する

オプション	長い形	働き
`-c`	`--omit-header`	`# file:` などの見出し行を出さない
`-e`	`--all-effective`	`#effective` の注記を、削られていない行にも付ける
`-E`	`--no-effective`	`#effective` の注記を出さない
`-s`	`--skip-base`	基本エントリだけの対象（拡張 ACL を持たない対象）を飛ばす
`-R`	`--recursive`	配下を再帰的にたどる
`-t`	`--tabular`	表形式で出力する
`-n`	`--numeric`	ユーザー名・グループ名ではなく数値の ID で表示する
`-p`	`--absolute-names`	先頭のスラッシュを取り除かずに出力する

`getfacl` は**読み出し専用**で、エントリを与える働きは持ちません。ACL を設定するのは `setfacl` です。

1-8 拡張属性 —— getfattr / setfattr と4つの名前空間

拡張属性は、**ファイルに「名前と値の組」を後から結び付けられる汎用の入れ物**です。名前は `名前空間.名前` という形をしていて、名前空間ごとに扱いが違います。

名前空間	誰が使うか	一般ユーザーが自由に読み書きできるか	カーネルのアクセス判定に使われるか
`user`	**アプリケーションと利用者**が自由に使うメタデータの置き場	**できる**（対象ファイルへの通常の権限があればよい）	**使われない**
`trusted`	特権を持つプロセス向け	できない	—
`security`	**カーネルのセキュリティ機構**が使う領域	できない	機構側が使う
`system`	カーネルが使う領域。**POSIX ACL の格納先**	できない（ACL は `setfacl` 経由で扱う）	**使われる**

`user` **名前空間の性格を押さえておいてください。**ここはアプリケーションが自由に使えるメタデータの置き場で、**カーネルのアクセス判定にはいっさい参照されません。**「機密区分のラベルを付けたい」「業務上の分類を記録したい」といった、権限そのものではないメタデータの記録先として向いています。逆に、**ここに何を書いてもアクセス制御にはなりません。**

```
# 機密区分のラベルを user 名前空間の拡張属性として付ける
setfattr -n user.classification -v confidential report.xlsx

# 付けたラベルを読み出す
getfattr -n user.classification report.xlsx
```

getfattr の既定は user 名前空間だけ —— ACL が出てこない理由

`getfattr` には**属性名の照合パターン**があり、**既定値は** `^user\.` です。つまり `getfattr -d` で表示されるのは **user 名前空間の属性だけ**です。

POSIX ACL は `system.posix_acl_access` / `system.posix_acl_default` という **system 名前空間の属性として格納されている**ので、**既定の照合パターンから外れ、何も表示されません。** 拡張 ACL が設定されているファイルに `getfattr -d` を打って何も出ないのは、属性が無いからではなく、**表示範囲がそもそも違う**からです。

表示範囲を変えるのが `-m`（`--match`）で、ここに別の正規表現パターンを渡します。`-m -` **を指定するとすべての名前空間が対象**になります。

```
# 既定（^user\. のみ）—— ACL は出てこない
getfattr -d report.txt

# すべての名前空間を対象にする → ACL を保持する属性も見える
getfattr -d -m - report.txt
# file: report.txt
system.posix_acl_access=0sAgAAAAEABgD/////AgAGAOgDAAAEAAYA...
```

ただし、こうして見えた値は**符号化された形**で、そのままでは読めません。**ACL の確認と設定は** `getfacl` **と** `setfacl` **で行う** —— ここが `getfattr` ／ `setfattr` との役割の分かれ目です。

getfattr のオプション

オプション	長い形	働き
`-n`	`--name`	**属性名をひとつ指定して、その値を取り出す**
`-d`	`--dump`	**一致したすべての属性の値を出力する**（名前の一覧だけではなく値まで出す）
`-e`	`--encoding`	値の出力形式を `text`・`hex`・`base64` から選ぶ
`-m`	`--match`	**表示対象とする属性名の正規表現パターンを渡す**。既定は `^user\.`、`-` ですべて
`-h`	`--no-dereference`	シンボリックリンクをたどらず、リンク自体を対象にする
`-R`	`--recursive`	配下を再帰的にたどる
`-L` / `-P`	`--logical` / `--physical`	再帰時にディレクトリへのシンボリックリンクをたどる／たどらない
`--only-values`	—	符号化せずに生の値だけを出す

`-n` と `-m` の違いは頻出です。 `-n` は**属性名をひとつ名指しする**指定で、名前空間をまたいで表示範囲を広げるものではありません。名前空間の範囲を変えるのは `-m` です。`-e` は出力の符号化形式、`-R` は再帰、`-h` はシンボリックリンクの扱いで、いずれも表示対象の名前空間は変えません。

setfattr のオプション

オプション	長い形	働き
`-n`	`--name`	設定する拡張属性の**名前**を指定する
`-v`	`--value`	設定する**値**を指定する
`-x`	`--remove`	指定した拡張属性を**削除する**
`-h`	`--no-dereference`	シンボリックリンクをたどらない
`--restore`	—	`getfattr --dump` の出力形式から拡張属性を復元する

値は、二重引用符で囲めばテキスト、`0x` で始めれば16進数、`0s` で始めればbase64 として解釈されます。

setfacl と setfattr の役割分担

POSIX ACL は実体としては拡張属性の一種ですが、`setfacl` はその ACL 用の属性を ACL の文法だけで扱う専用の窓口です。 任意の名前と値の組を書き込む口にはなっていません。

	setfacl / getfacl	setfattr / getfattr
扱う対象	**POSIX ACL 専用**（決められた名前の属性）	**任意の拡張属性**（名前空間と名前を自分で指定）
入力の文法	`u:alice:rwx` のような **ACL のエントリ書式**	`-n 名前 -v 値` の**名前と値の組**

	setfacl / getfacl	setfattr / getfattr
デフォルト ACL の扱い	`-d` で扱える	**専用の書式を持たない**
用途	権限の付与・確認	権限ではないメタデータの記録・確認

「ACL とは別に、ファイルへ独自の名前と値の組を拡張属性として直接書き込みたい」という要件で使うのは **setfattr** です。`setfacl` は ACL 専用、`getfattr` は読み出し専用、`getfacl` は ACL の表示専用で、どれも書き込みの窓口にはなりません。

1-9 ACL を意識するコマンド —— ls / cp / mv / rsync / tar

ls —— `+` は「ACL がある」だけを示す

`ls -l` は、パーミッション文字列の末尾に `+` を表示することがあります。表示される条件は次の**どちらか**です。

- **基本の3エントリを超えるアクセス ACL**（＝拡張 ACL）を持っている
- **デフォルト ACL を持っている**（ディレクトリの場合）

```
drwxrwx---+  2 root root 4096 Aug 27 10:00 proj
-rw-rw----+  1 kenta staff 120 Aug 27 10:01 report.txt
```

つまり **`+` だけでは、いま効いている権限が拡張されているのか、以後の継承だけが設定されているのかを区別できません。** 棚卸しでは `+` の付いた対象を拾ったあと、**`getfacl` で中身を確認する工程が必ず要ります。**

誤り肢の材料になるのは次の3点です。

- **「名前付きユーザーのエントリを持つ対象だけが `+` になる」は誤り** —— 名前付きグループのエントリだけでも付きますし、デフォルト ACL しか持たないディレクトリにも付きます
- **「マスクを持つ対象だけが `+` になる」も条件として成り立たない** —— 名前付きエントリを持つ ACL には必ずマスクが伴うので、この2つを区別できません
- **`+` が示すのは ACL の有無であって、拡張属性の有無ではありません** —— `user` 名前空間の属性だけを付けたファイルに `+` は付きません

もうひとつ、`ls -l` のグループ欄の意味も要注意です。マスクを持つ ACL では、**`ls -l` のグループ欄に出るのはマスクの値**であって、名前付きエントリへ実際に設定した権限ではありません（1-4 のパーミッションビットの対応そのものです）。`ls -l` で `rwx` に見えていても、名前付きエントリの実効権限はマスクで頭打ちになります。

cp と mv —— 「作り直すか」で決まる

観点	cp（オプションなし）	mv（同一ファイルシステム内）
何をしているか	**コピー先に新しくファイルを作る**	ファイルの実体を作り直さず、**参照先を付け替える**
元の ACL	**引き継がれない**	**そのまま保たれる**（マスクも含めて丸ごと）
移動先／コピー先のデフォルト ACL	**適用される**（新規作成だから）	**適用されない**（新規作成ではないから）
ACL を保つには	`-p` ／ `--preserve=all` ／ `-a` を付ける	何もしなくても保たれる

この対比が、この副主題で最も問われる組のひとつです。デフォルト ACL は**オブジェクトが新しく作られるときにだけ**効くので、`cp` には効き、`mv` には効きません。

- `cp` は既定では ACL を引き継ぎません。オプションなしでコピーすれば、**コピー先ディレクトリのデフォルト ACL に従った ACL** が付きます。`cp -p` は `--preserve=mode,ownership,timestamps` と同じで、この `mode` に **ACL と拡張属性の権限が含まれます。** `-a` は `-dR --preserve=all` と同じです
- ACL の扱いが**同一ファイルシステムかどうかで切り替わることはありません。** `cp` はオプションの有無で決まります
- **「`cp` が ACL を読み出す権限を持たないから」という説明も成り立ちません。** ACL の読み出しは所有者や特権の有無で決まる話で、**マスクエントリだけが選択的に再構築されて残りが初期化される、といった動作も存在しません**
- `mv` では、元の ACL と移動先のデフォルト ACL が**統合される仕組みも、マスクだけが再計算される仕組みもありません。**丸ごとそのまま残ります

運用上の含意も押さえておいてください。**共有ディレクトリの権限設計をデフォルト ACL で組んでいても、他所から `mv` で持ち込まれたファイルには移動先の設計が反映されません。**想定と食い違った ACL がそのまま残ります。

rsync —— `-a` に ACL と拡張属性は入っていない

`rsync` の `-a`（`--archive`）は **`-rlptgoD` と同じ**で、再帰・シンボリックリンク・パーミッション・タイムスタンプ・グループ・所有者・デバイスファイルをまとめて指定する省略形です。

`-a` には ACL（`-A`）も拡張属性（`-X`）も含まれません。マニュアルにも、`-a` は ACL・xattr・atime・crtime・ハードリンクの保全を含まない、と明記されて

います。

オプション	長い形	働き
`-A`	`--acls`	**転送先の ACL を転送元と同じにする**。この指定は `--perms` **を含む**
`-X`	`--xattrs`	**転送先の拡張属性を転送元と同じにする**
`-p`	`--perms`	転送先のパーミッションを転送元と同じにする

```
# ACL と拡張属性まで含めて同期する
rsync -avAX /srv/share/ backup:/srv/share/
```

- **「`-a` に ACL は含まれるが拡張属性は含まれない」は誤りです。**どちらも含まれません
- **転送方式（差分か全量か）で ACL の扱いが変わることはありません。**`--checksum` を付けても ACL は運ばれません。ACL が運ばれるかどうかは `-A` の有無だけで決まります
- **`-p` が ACL の転送を打ち消すことはありません。**むしろ逆で、`-A` のほうが `-p` を含みます

なお、**「ACL を保ったまま、前回から変わった部分だけをリモートホストへ送る」という要件に当たるコマンドは `rsync` です。**`cp` はローカルの複製で差分判定もリモート転送も持たず、`tar` は ACL を保存できても毎回アーカイブ全体を作り直す方式、`mv` は移動と改名、`setfacl` は ACL の設定であって転送を扱いません。

tar —— ACL は「作るとき」に記録させる

`tar` で ACL を保全するには、**アーカイブを作る時点で `--acls` を指定して ACL を記録させておく必要があります。**拡張属性は `--xattrs` で別に指定します。

オプション	働き
`--acls` / `--no-acls`	POSIX ACL の対応を**有効**／**無効**にする
`--xattrs` / `--no-xattrs`	拡張属性の対応を有効／無効にする
`--xattrs-include=` / `--xattrs-exclude=`	対象とする拡張属性の名前を**グロブパターンで絞る**（例: `user.*`）
`-p` / `--preserve-permissions` / `--same-permissions`	展開時に、**アーカイブに記録されている**パーミッションを反映させる

```
# 作成時に ACL と拡張属性を記録させる
tar --acls --xattrs -cvzf backup.tar.gz /srv/share

# 展開時にも指定する
tar --acls --xattrs -xvzf backup.tar.gz
```

押さえどころは「記録されていないものは復元できない」ことです。作成時に `--acls` を付けていないアーカイブからは、展開側でどんな指定をしても ACL は取り出せません。

- **`--same-permissions` はアーカイブに記録されたパーミッションを反映させるオプション**であって、記録されていない ACL を作り出すことはできません

- **ACL を記録できるかどうかは圧縮の有無とは無関係です。**`gzip` を通しても ACL は記録できます。決めるのは `--acls` の有無です
- `tar` **が ACL を扱うのは** `--acls` **です。**`--xattrs-include='user.*'` のように `user` 名前空間の属性を対象にする指定では、ACL（system 名前空間）は復元されません
- 既定で ACL を扱うかどうかはディストリビューションによって差があるため、**保全したいなら明示的に指定するのが確実です**

`rsync` **が** `-A` **と** `-X` **を都度指定するのと同じ構図で、**`tar` **でも ACL と拡張属性は別々に、しかも明示的に指定する** —— この対応関係で覚えておくと取り違えにくくなります。

1-10 ファイル共有での実装差異 —— NFSv4 と Samba

この項目で問われるのは**「モデルが違う」という実装差異の理解**であって、NFSv4 ACL や Windows ACL の専用の操作手順ではありません。

NFSv4 ACL は別系統のモデル

観点	POSIX ACL（Linux）	NFSv4 ACL
エントリの種類	許可を与えるエントリだけ	**許可（Allow）と拒否（Deny）を明示するエントリ**（ほかに監査・警報の種別もある）
評価の仕方	所有者→名前付きユーザー→グループ→other の順に**一致したクラスで判定**	**ACE を並び順に評価する**
マスク	**ある**（`ACL_MASK` が実効権限の上限を決める）	**無い**（マスクに相当する概念が存在しない）

観点	POSIX ACL（Linux）	NFSv4 ACL
継承の担い手	**ディレクトリのデフォルト ACL**	**各エントリが持つ継承フラグ**（ファイル継承・ディレクトリ継承・継承のみ・伝播しない など）
利用者の識別	ローカルの UID／GID	**名前@ドメイン の形の文字列**（`OWNER@`・`GROUP@`・`EVERYONE@` という特別な指定もある）

別のモデルなので、サーバ側の POSIX ACL とクライアントから見える権限は一対一に写りません。対応付け（マッピング）の過程で表現しきれない部分が必ず出ます。設計上の留意点は、まさにこの**「権限の対応付けが必要になる」**という点です。

誤り肢としてよく置かれるのは次の3つです。

- **「NFSv4 の ACL は POSIX ACL と同じモデルで、マスクもそのまま転送される」は誤り。**マスクに相当する概念が NFSv4 側にありません
- **「NFSv4 では継承がエントリの継承フラグに置き換わるので、ディレクトリ側の設定は不要でファイル単位の設定だけで継承が成立する」は誤り。**継承フラグを各エントリが持つのは NFSv4 側の仕組みですが、**サーバ側で POSIX ACL を用いる限り、継承を担うのはディレクトリのデフォルト ACL です。**ファイル単位の設定だけでは継承は成立しません
- **「NFSv4 は数値の UID と GID だけで識別するので解釈が一致する」は誤り。**NFSv4 は**名前の形式で利用者を識別する**設計なので、サーバとクライアントで識別子の解釈をそろえる仕組み（ID マッピング）が別途必要になります

Samba は Windows の ACL を POSIX ACL へ対応付ける

Samba は、Windows クライアントが扱う ACL を**サーバ側の POSIX ACL へ対応付けて公開します。** Windows 側の ACL も、許可と拒否のエントリを持ち、継承フラグを各エントリが持つモデルです。

Windows でいう「継承」に相当する働きを、サーバ側で担うのはディレクトリのデフォルト ACL です。

```
# Samba 共有の中に新規作成されるものへ、開発グループの読み書きを自動で付ける
setfacl -m g:dev:rwx /srv/samba/proj      # 共有ディレクトリ自身への権限
setfacl -d -m g:dev:rwx /srv/samba/proj   # これから作られるものへの雛形
```

ここでも誤り肢の作りは同じです。**アクセス ACL に名前付きグループを足しても、配下の対象へ随時反映される仕組みはありません。マスクを `rwx` にしても、既存のエントリの実効権限の上限が上がるだけで、新規作成される対象へエントリを引き継ぐ働きはありません。`user` 名前空間の拡張属性に継承ルールを書いても、Samba がそれを解釈して権限の継承を成立させることはありません。**

設計上の留意点として、**両者のモデルが一致しないため、マスクによって実効権限が絞られると、Windows 側の表示と実際のアクセス可否がずれることがあります。** Samba には共有単位で ACL の継承やパーミッションの扱いを調整するパラメータがありますが、実体としての継承を担っているのはサーバ側のデフォルト ACL です。

NFSv4 でも Samba でも、雛形はサーバ側に置く

NFSv4 と Samba の双方で同じディレクトリを公開している場合でも、ファイルの生成はサーバ側のファイルシステムで行われます。したがって、「以後このディレクトリ配下に作られるものへ自動的に権限を付けたい」という要件で使うのは、**サーバ側で `setfacl` にデフォルト ACL を設定する**ことです。

`getfacl` は読み出し側、`setfattr` は名前空間と属性名を指定して拡張属性そのものを書き込むコマンドで、ACL 専用の書式やデフォルトの扱いを備えていません。`getfattr` は読み出し側で、既定の照合パターンでは `user` 名前空間だけが対象です。`mv` は名前や置き場所を変える操作で、同一ファイルシステム内なら設定済みの ACL はそのまま付いて回りますが、**新しいエントリを与える手段ではありません。**

第1章の混同しやすい対の概念

観点	アクセス ACL（ACL Entry）	デフォルト ACL（Default ACL）
効く相手	**そのオブジェクト自身**	**これから新しく作られるオブジェクト**
設定できる対象	ファイルとディレクトリ	**ディレクトリのみ**
アクセス判定	**使われる**	**使われない**（雛形として写されるだけ）
setfacl の指定	既定（`-m` など）	`-d`（`--default`）または `d:` 接頭辞
削除	`-b`（拡張エントリを全削除）	`-k`（`--remove-default`）

観点	アクセス ACL（ACL Entry）	デフォルト ACL（Default ACL）
格納先の属性	`system.posix_acl_access`	`system.posix_acl_default`

観点	setfacl -R	setfacl -d
効く相手	**実行時点で存在する**ファイルとディレクトリ	**実行後に作られる**ファイルとディレクトリ
既存ファイル	対象	対象外
将来のファイル	**対象外**	対象

観点	マスクの影響を受けるエントリ	受けないエントリ
該当	`user:名前:`（名前付きユーザー）／`group:名前:`（名前付きグループ）／`group::`（所有グループ）	**`user::`（所有者）**／**`other::`**
効く権限	エントリ ∧ マスク（削られると `#effective:` が付く）	エントリの権限がそのまま

観点	マスクを持つ ACL	マスクを持たない ACL
権限ビットの group 部分が対応する先	**`mask::`**	**`group::`（所有グループ）**
`chmod g-w` が絞るもの	**マスク**（名前付きエントリはそのまま残るが実効だけ落ちる）	所有グループのエントリ
`ls -l` のグループ欄	**マスクの値**	所有グループの権限

観点	setfacl -n（ --no-mask ）	setfacl --mask
マスクの再計算	**しない**	**明示的に与えられていても再計算する**
起きること	書いた権限が既存の狭いマスクで削られ `#effective` が付く	マスクが必要な値まで引き上げられる

観点	cp（オプションなし）	mv（同一ファイルシステム内）
実体	**新しく作る**	**作り直さない**（参照先の付け替え）
元の ACL	引き継がれない	**そのまま保たれる**
先のデフォルト ACL	**適用される**	適用されない
ACL を保つ指定	`-p` / `--preserve=all` / `-a`	不要

観点	getfacl / setfacl	getfattr / setfattr
対象	**POSIX ACL 専用**	**任意の拡張属性**
既定の表示範囲	ACL の全エントリ	`user` **名前空間のみ**（`^user\.`）
ACL は見えるか	見える	**既定では見えない**（`-m -` で見えるが符号化された値）
用途	権限の付与・確認	権限ではないメタデータの記録・確認

観点	user 名前空間	system 名前空間	security 名前空間
誰が使うか	アプリケーション・利用者	カーネル（**POSIX ACL の格納先**）	**カーネルのセキュリティ機構**
一般ユーザーが自由に読み書き	**できる**	できない	**できない**
アクセス判定に使われるか	**使われない**	使われる	機構側が使う

観点	setfacl --restore	setfacl -M / --set-file
読むもの	**getfacl -R の出力**（`# file:` の見出し付き）	**エントリの並びだけ**
対象ごとの区別	**できる**（見出しのパスへ戻す）	できない（指定した対象へ一律適用）
既存 ACL	記録どおりに戻す	`-M` は変更、**`--set-file` は置き換え**
注意	退避時と復元時でカレントディレクトリを揃える	ツリー全体の復元には使えない

観点	rsync -A	rsync -X	tar --acls	tar --xattrs
運ぶもの	**ACL**	**拡張属性**	**ACL**	**拡張属性**
`-a` に含まれるか	**含まれない**	**含まれない**	—	—
備考	**`--perms` を含む**	—	**作成時に指定が要る**	`--xattrs-include=` で絞れる

観点	POSIX ACL	NFSv4 ACL / Windows ACL
拒否の明示	**できない**（許可を与えるだけ）	**できる**（Deny のエントリを持つ）
マスク	**ある**	**無い**
継承	**ディレクトリのデフォルト ACL**	**各エントリの継承フラグ**
識別	UID／GID	**`名前@ドメイン` の形の文字列**（NFSv4）
対応付け	—	**一対一に写らないためマッピングが要る**

第1章の入力式で問われる綴り

何を答えるか	綴り	補足
新規作成されるものへ権限を継承させるため、デフォルト ACL として設定する `setfacl` のオプション	**`-d`**（`--default`）	エントリ側に **`d:`** の接頭辞を書く形も同じ意味になる
ACL 専用ではなく、拡張属性そのものへ名前と値の組を書き込むコマンド	**`setfattr`**	`/usr/bin/setfattr`
`getfattr` で、表示対象とする名前空間の正規表現パターンを渡すオプション	**`-m`**（`--match`）	既定のパターンは **`^user\.`**。**`-m -`** ですべての名前空間が対象
ACL を保ったまま、前回から変わった部分だけをリモートホストへ送るコマンド	**`rsync`**	`/usr/bin/rsync`。ACL は **`-A`**、拡張属性は **`-X`**

何を答えるか	綴り	補足
有効権限マスクの再計算を抑止する `setfacl` のオプション	**`-n`**（`--no-mask`）	これを付けると `#effective:` で権限が削られる
親ディレクトリに既定の ACL エントリを与えるコマンド	**`setfacl`**	`/usr/bin/setfacl`・`/bin/setfacl`。`-m` が追加・変更、`-d` がデフォルト側

紛らわしい綴りの区別（誤答として置かれるもの）

綴り	何をするものか
`setfacl -m`（`--modify`）	ACL エントリを**追加・変更する**という指定そのもの。アクセス ACL かデフォルト ACL かまでは決めない
`setfacl -R`（`--recursive`）	**既に存在する**ファイルとディレクトリを再帰的にたどる。これから作られるものには効かない
`setfacl -k`（`--remove-default`）	**デフォルト ACL を削除する**（設定する側ではない）
`setfacl -b`（`--remove-all`）	**拡張 ACL エントリをすべて削除**し、基本のパーミッションだけを残す
`setfacl --mask`	マスクを**必ず再計算させる**（`-n` とは逆向き）
`setfacl -x`（`--remove`）	指定した ACL エントリを**削除する**
`setfacl --test`	適用後の ACL を**表示するだけ**で、実際には変更しない
`setfacl -P`（`--physical`）	再帰処理でシンボリックリンクを**たどらない**

綴り	何をするものか
`getfattr -n`（`--name`）	属性名を**ひとつだけ名指し**して値を取り出す。名前空間の絞り込みではない
`getfattr -d`（`--dump`）	一致した属性の**値まで出力する**（名前の一覧だけを出すものではない）
`getfattr -e`（`--encoding`）	値を `text`・`hex`・`base64` の**どれで出力するか**を決める
`getfattr -h`（`--no-dereference`）	シンボリックリンクを**たどらない**
`getfattr -R`（`--recursive`）	配下をたどる。**表示対象の名前空間は変わらない**
`getfacl`	設定済みの ACL を**読み出して表示する側**。エントリを与える働きはない
`getfattr`	拡張属性を**読み出す側**。既定では `user` 名前空間だけが対象
`cp`	ローカルの複製。**リモート転送も差分判定も持たない**
`tar`	アーカイブの作成・展開。ACL は保存できるが、**差分を判定して送る対象を絞る働きは持たない**
`mv`	移動と改名。同一ファイルシステム内なら ACL はそのまま付いて回るが、**エントリを与える手段ではない**

第1章の試験ポイント

- **アクセス ACL とデフォルト ACL の入れ替え（①型）**：「アクセス ACL に足せば配下の新規ファイルへ引き継がれる」は誤りです。**継承を担うのはデ

フォルト ACL だけで、アクセス ACL はそのオブジェクト自身の可否を決めるだけです

- **`-R` と `-d` のすり替え（③型）**：「既存の権限は調整済み。**今後**作られるものに」という条件が出たら答えは **`-d`** です。`-R` は実行した時点で存在するものにしか届きません。逆に「すでに置かれているファイルに一括で」なら `-R` で、`-d` では届きません
- **デフォルト ACL をファイルに設定する（⑦型）**：「デフォルト ACL はファイル単体にも設定でき、開いたプロセスに適用される」は誤りです。**持てるのはディレクトリだけ**で、アクセス判定にも参加しません
- **継承の絞り込みを umask のせいにする（④型）**：デフォルト ACL がある場合、新規オブジェクトは**デフォルト ACL をアクセス ACL として受け継ぎ、作成時の mode に含まれない権限が取り除かれます。**umask が効くのはデフォルト ACL が無いときの経路です。`touch` で作ったファイルに実行権が残らないのは、mode に実行権が含まれないからです
- **サブディレクトリの継承（①型）**：新規ファイルはデフォルト ACL を**アクセス ACL として**受け取り、新規サブディレクトリは**アクセス ACL とデフォルト ACL の両方を**受け取ります
- **マスクの適用対象を取り違える（①型）**：`#effective:` が出たら原因は**マスク**です。**所有者（`user::`）と `other::` はマスクの対象外**なので、そこを広げても名前付きユーザーの実効権限は変わりません。**デフォルト ACL 側のマスクでもありません**（それは以後作られるものの初期値に関わります）
- **chmod がどこに効くかを取り違える（②型）**：拡張 ACL を持つファイルでは**権限ビットの group 部分がマスクに対応**します。`chmod g-w` は**マスクを絞る操作**であって、名前付きユーザーのエントリを削除するわけでも、所有グループの権限が上限になるわけでもありません。マスクを持たないファイルでは group 部分は所有グループのエントリに対応します

- **`-n`（`--no-mask`）の事故（⑤④型）**：「エントリに `rwx` と書いたのに `#effective:r--` になる」の原因の1つが `-n` です。`setfacl` は**既定ではマスクを必要な値まで引き上げます**。`--mask` はその逆で、明示指定があっても再計算させます
- **`ls` の `+` の意味（①③型）**：`+` が付くのは**拡張 ACL を持つ対象**と**デフォルト ACL を持つディレクトリ**の両方です。「名前付きユーザーのエントリを持つ対象だけ」「マスクを持つ対象だけ」は条件として成り立ちません。**拡張属性の有無を示すものでもありません**
- **`ls -l` のグループ欄（①型）**：マスクを持つ ACL では、グループ欄に出るのは**マスクの値**です。名前付きエントリに設定した権限が並ぶわけではありません
- **`cp` と `mv` の入れ替え（③型）：新しく作る `cp` にはコピー先のデフォルト ACL が効き、作り直さない `mv` には効きません。** `cp` で ACL を保つには `-p` ／ `--preserve=all` ／ `-a` が要ります。「同一ファイルシステムかどうかで `cp` の挙動が変わる」は誤りで、決めるのはオプションです
- **`rsync -a` に ACL が入っていると思い込む（④⑦型）**：`-a` は `-rlptgoD` で、**ACL（`-A`）も拡張属性（`-X`）も含みません。**「`-a` に ACL は含まれるが xattr は含まれない」も誤りです。`--checksum` などの転送方式は ACL の扱いを変えません。`-p` が `-A` を打ち消すこともなく、**`-A` のほうが `--perms` を含みます**
- **`tar` の指定時期（⑤⑥型）**：ACL は**作成時に `--acls` を付けて記録させて**おかないと、展開時に何を指定しても復元できません。**`--same-permissions` は記録済みのものを反映するだけ**で、記録の無い ACL を作り出せません。圧縮の有無は無関係で、`--xattrs-include='user.*'` では ACL（system 名前空間）は戻りません

- **退避と復元の道具違い（①型）**：ツリー全体の ACL を戻せるのは `getfacl -R` + `setfacl --restore` の組だけです。`-M` と `--set-file` **はエントリの並びしか読まない**ので対象ごとに戻せず、`--set-file` は既存 ACL を置き換えます。`getfattr -R -d` では ACL を保持する属性そのものが退避されません（既定が `^user\.` のため）
- `getfattr` **で ACL が見えない理由（②型）**：既定の照合パターンが `^user\.` で、POSIX ACL は `system` **名前空間**にあるためです。「値を持たないから」でも「ACL は inode に直接あって拡張属性ではないから」でもありません。`-m -` **で見えますが値は符号化されている**ので、確認と設定は `getfacl` ／ `setfacl` を使います
- **ラベル付けの置き場（②型）**：一般ユーザーの権限で自由に付け外しできるのは `user` **名前空間**だけです。`security` はカーネルのセキュリティ機構の領域、`system.posix_acl_default` は POSIX ACL の格納先で、任意の文字列を業務ラベルとして書き込む先ではありません。**「権限を与えたくない相手の ACL エントリを目印にする」は、実際のアクセス権が設計と食い違うため成立しません**
- **NFSv4 ACL を POSIX ACL と同じモデルとみなす（①型）**：NFSv4 側には**拒否を明示するエントリがあり、マスクに相当する概念がありません。**継承は各エントリの継承フラグが担い、利用者は**名前の形式**で識別されるため ID の対応付けが別途要ります。「数値の UID／GID だけで識別するから解釈が一致する」は誤りです
- **Samba の継承をアクセス ACL やマスクで作ろうとする（①型）**：Windows の継承に相当するのは**サーバ側ディレクトリのデフォルト ACL** です。アクセス ACL に足しても配下へ随時反映されませんし、マスクを `rwx` にしても新規作成物へエントリは引き継がれません。`user` **名前空間の拡張属性に継承ルールを書いても Samba は解釈しません**

- **公開プロトコルを問わず雛形はサーバ側（②型）**：NFSv4 でも Samba でも**ファイルの生成はサーバ側のファイルシステムで起きる**ので、継承の設計は `setfacl -d` でサーバ側に置きます

サンプルはここまでです

続き(第2章〜巻末)は、LinuC 3SS 問題集のご購入で、暗記用ノート・頻出度別ノートと合わせて3点すべてダウンロードできます。

LinuC レベル3 3SS試験 学習用テキスト(無料サンプル)

版

2026年7月版(LPI-Japan公式 LinuC レベル3 3SS試験(LinuC-3SS Version 1.0) 出題範囲 準拠)

発行

IT資格道場(https://www.shikaku-doujou.com)

お問い合わせ

info@shikaku-doujou.com