Korte uitleg: Wie mag wat met welk bestand

Zoals in *Korte uitleg: Directories, meta-data van bestanden, inodes* beschreven zit er bij de meta-data van een bestand in de inode van dat bestand een aantal bitjes dat aangeeft wie toestemming heeft om iets met dat bestand te doen. Dus moeten we weten wie die "wie" is die misschien iets mag doen (categorieën). En wat de handelingen zijn die met het bestand mogen worden uitgevoerd (permissies).

Categorieën

Bij ieder bestand horen drie categorieën. Voor ieder van die categorieën wordt apart bepaald welke permissies zij hebben voor processen:

1. de eigenaar (een `uid`), deze permissies gelden als de proceseigenaar dezelfde uid heeft als de bestandseigenaar (korte naam voor deze permissie: `u`);
2. een groep (een `gid`), als de proceseigenaar niet onder 1 valt, maar wel tot deze gid behoort, gelden deze permissies (aangeduid met `g`);
3. alle andere processen, die niet onder 1 en 2 vallen hebben deze permissies (aangeduid met `o`).

Permissies

Er zijn drie permissies:

1. permissie om te lezen uit het bestand (aangeduid met `r`);
2. permissie om te schrijven naar het bestand (`w`);
3. permissie om een bestand uit te voeren (`x`), dat heeft natuurlijk alleen zin als het een uitvoerbaar bestand (binary executable of script) is, je kan hier niet iets "executable mee maken" (deze uitdrukking wordt vaak foutief gebruikt).

Bij directories werkt dit een beetje anders, bovendien moet je even goed nadenken wat lezen en schrijven bij een directory inhoudt. Daarom praten we verderop apart over directories.

Meestal worden de permissies opgeschreven in een rij achter elkaar met een `-` als de permissie afstaat en wel in de volgorde `rwx` en dat drie keer achter elkaar in de volgorde behorende bij `ugo`. Zie hier:

```
henk@boven:~/test> ls -ln | grep -v '^d'
totaal 20
-rw-r--r-- 1 500 500    7  8 jul 15:53 file
-rwxr--r-- 1 500 500   49 13 mrt 10:39 script
-rw------- 1 500 500    3 31 mrt 19:29 spsp
henk@boven:~/test>
```

De eerste `-` op iedere regel betekent "gewoon bestand". We zien hier de `uid` en

`gid` van ieder bestand i.p.v. de username en groupname (ze zijn allebei `500`, maar dat is toeval).
Het bestand `file` kan gelezen worden door `u`, `g` en `o`, door iedereen dus. Maar het mag alleen beschreven worden door een proces met `uid 500` als proceseigenaar.
Het bestand `script` heeft hetzelfde, maar nu mag een proces met eigenaar `500` `script` ook uitvoeren.
Het bestand `spsp` mag alleen vanuit processen met eigenaar `500` gelezen en geschreven worden.

Meestal staan de meeste permissies aan voor `u`, eventueel wat minder voor `g` en de minste voor `o`. Maar dat hoeft niet en je kunt leuke, niet allemaal zinvolle, combinaties maken:

```
henk@boven:~/test> ls -ln | grep -v '^d'
totaal 20
-rw-r--r-- 1 500 500    7  8 jul 15:53 file
-rwxr--r-- 1 500 500   49 13 mrt 10:39 script
-rw------- 1 500 500    3 31 mrt 19:29 spsp
henk@boven:~/test> cat file
inhoud
henk@boven:~/test> chmod u-r file
henk@boven:~/test> ls -ln | grep -v '^d'
totaal 20
--w-r--r-- 1 500 500    7  8 jul 15:53 file
-rwxr--r-- 1 500 500   49 13 mrt 10:39 script
-rw------- 1 500 500    3 31 mrt 19:29 spsp
henk@boven:~/test> cat file
cat: file: Toegang geweigerd
henk@boven:~/test> echo "nieuwe inhoud" >>file
henk@boven:~/test> cat file
cat: file: Toegang geweigerd
henk@boven:~/test> ls -ln | grep -v '^d'
totaal 20
--w-r--r-- 1 500 500   21  8 jul 16:00 file
-rwxr--r-- 1 500 500   49 13 mrt 10:39 script
-rw------- 1 500 500    3 31 mrt 19:29 spsp
henk@boven:~/test>
```

We kunnen nog steeds niet lezen, maar aan de `ls` uitvoer is zien we dat `file` nu 21 tekens groot is en dat was 7. En jawel:

```
henk@boven:~/test> chmod u+r file
henk@boven:~/test> cat file
inhoud
nieuwe inhoud
henk@boven:~/test>
```

En dan nu voor directories:

1. Het `r` bit maakt lezen van de directory mogelijk. Dat wil dus zeggen dat je kunt zien wat er in de directory staat en dat is de lijst van bestandsnamen.
2. Het `w` bit maakt schrijven in de directory mogelijk en dat betekent het aanmaken, verwijderen en hernoemen van bestanden in de directory.

3. Het `x` bit geeft toestemming tot het gebruik van de inodes via de directory. En dat betekent dat alleen dan de meta-data van de bestanden in de directory ter beschikking zijn om gelezen en gewijzigd te worden. Tot die meta-data behoort ook de plek op schijf. Je kan dus niet bij de gegevens als je niet bij de inode kunt!

Denk er even over na en je zult begrijpen dat:

- De `r` en `x` permissies van alle directories in een pad naar een bestand voor een proces aan moeten staan om gebruik van dat bestand mogelijk te maken.
- Je een bestand kan verwijderen ook al kan je dat bestand niet lezen. Voor verwijderen is `w` permissie voor de directory waarin het bestand staat nodig, geen permissie voor het bestand zelf.

```
henk@boven:~/test/wdir> l
totaal 12
drwxr-xr-x 2 henk   wij 4096  8 jul 18:27 ./
drwxr-xr-x 5 henk   wij 4096  8 jul 18:26 ../
-rw-r----- 1 wwwrun www  796  8 jul 18:27 index.html
henk@boven:~/test/wdir> rm index.html
rm: normaal bestand ‘index.html’ (schrijfbeveiligd) verwijderen? j
henk@boven:~/test/wdir> l
totaal 8
drwxr-xr-x 2 henk wij 4096  8 jul 18:29 ./
drwxr-xr-x 5 henk wij 4096  8 jul 18:26 ../
henk@boven:~/test/wdir>
```

We zien hierboven user `henk`, die in een directory kijkt waar `henk` de eigenaar is. Daar zit een bestand waar hij niet de eigenaar van is. Aangezien de wereld/anderen niet mogen lezen kan `henk` de inhoud niet zien. Dan probeert `henk` deze indringer te verwijderen. Het `rm` programma is nog zo vriendelijk om te waarschuwen, maar het doet het wel omdat `henk` alle rechten heeft in de directory `.` .

```
henk@boven:~/test> chmod a-x bestanden/
henk@boven:~/test> ls -ld bestanden/
drw-r--r-- 2 henk wij 4096  8 jul 18:26 bestanden/
henk@boven:~/test> ls -l bestanden/
ls: kan geen toegang krijgen tot bestanden/index.html: Toegang geweigerd
ls: kan geen toegang krijgen tot bestanden/verf.jpeg: Toegang geweigerd
ls: kan geen toegang krijgen tot bestanden/WARNING_README.txt: Toegang geweigerd
ls: kan geen toegang krijgen tot bestanden/2noot: Toegang geweigerd
ls: kan geen toegang krijgen tot bestanden/ASH.html: Toegang geweigerd
ls: kan geen toegang krijgen tot bestanden/verf.gif: Toegang geweigerd
totaal 0
-????????? ? ? ? ?            ? 2noot
-????????? ? ? ? ?            ? ASH.html
-????????? ? ? ? ?            ? index.html
-????????? ? ? ? ?            ? verf.gif
-????????? ? ? ? ?            ? verf.jpeg
-????????? ? ? ? ?            ? WARNING_README.txt
henk@boven:~/test> chmod u+x,u-r bestanden/
henk@boven:~/test> ls -l bestanden/
ls: kan map bestanden/ niet openen: Toegang geweigerd
```

```
henk@boven:~/test> ls -l bestanden/ASH.html
-rw-r--r-- 1 henk wij 3964 10 mei 20:42 bestanden/ASH.html
henk@boven:~/test>
```

We halen het `x` bit weg van de directory `bestanden`. Het gevolg zie je eronder, We zien wel welke bestanden er in `bestanden` zitten, maar verder krijgen we geen gegevens.
Dan het `x` bit weer aan en het `r` bit eraf. Gevolg, geen toegang. Logisch. Maar als we toevallig de naam van een bestand in de directory weten, kunnen we er toch bijkomen.

Verrassing?

Dus als je ooit een "permission problem" krijgt, vraag je dan atijd af wie de user en groep van het proces zijn dat fout gaat, en hoe dat klopt met de permissie bitjes van het bestand. En van de directory daarboven. En daarboven .....

Nog drie bitjes

Er zijn nog drie bitjes voor alle categorieën van betekenis zijn als ze aanstaan, onafhankelijk van `u`, `g` of `o`:

1. SUID (setuid of set userID). Dit bit heeft alleen betekenis als het bestand uitvoerbaar is. Als het programma wordt gestart, wordt de eigenaar (`uid`) van het bestand de eigenaar van het proces i.p.v. de `uid` van het ouder proces.

   ```
   henk@boven:~/test> ls -l $(which su)
   -rwsr-xr-x 1 root root 44256 23 jan 14:04 /usr/bin/su
   henk@boven:~/test>
   ```

   Op de plaats van de `x` voor `u` staat nu een `s`. Die `x` denk je erbij, want zonder `x` heeft die `s` geen zin. Als `x` toch afstaat wordt er een `S` getoond.
2. SGID (setgid of set group ID). Bij starten van het bestand krijgt het proces deze `gid` i.p.v. de `gid` van het aanroepende proces. Als dit op een directory staat krijgen nieuwe bestanden/directories niet als group de default group van de user die dat doet, maar dezelfde group als die van de directory. Wordt ook getoond met `s` of `S`.

   ```
   henk@boven:~/test> ls -ld bestanden/
   d-wxr-Sr-- 2 henk wij 4096  8 jul 18:26 bestanden/
   henk@boven:~/test>
   ```

3. Het sticky bit of the restricted deletion flag. Het sticky bit op een uitvoerbaar bestand heeft in Linux geen betekenis. Als dit bit bij een directory hoort heet het "restricted deletion flag". Het betekent dat een proces alleen een bestand in die directory mag verwijderen of hernoemen als de proceseigenaar ook eigenaar is van de directory, of van het te verwijderen/hernoemen bestand. Het grote voorbeeld is:

```
henk@boven:~/test> ls -ld /tmp
drwxrwxrwt 19 root root 4096  8 jul 20:00 /tmp
henk@boven:~/test>
```

Dus ondanks dat iedereen bestanden in `/tmp` mag zetten, mag je alleen je eigen bestanden weer weggooien.Dit wordt aangegeven met `t` of `T`.

Octale notatie

We gebruiken tot nu toe de notatie zoals die door tools zoals `ls` en `chmod` gebruikt wordt: `rwxr-x--x`. Omdat het bitjes zijn kun je dat ook noteren als `111101001`, maar dat doet niemand. Wel gebruikelijk is de octale notatie daarvan: `751`. Het tool `chmod` kan daar ook mee werken. Soms is het één handig, soms het ander.

umask

Het `umask` is een masker van bitjes dat staat in de omgeving van een proces. Het wordt dus geërfd door de kind processen. Als een proces aan de Kernel vraagt om een nieuw bestand te creëren vraagt het daarbij ook om de permissie bitjes op een bepaalde waarde te zetten. De Kernel maskeert deze rij bitjes nu met de `umask`. Je kan dus voorkomen dat bepaalde bits gezet worden door je `umask` goed te kiezen. Je kan er geen bitjes bij zetten.

Dus als een proces een directory wil aanmaken en daarbij voorstelt om dat `rwxr-xr-x/755` te doen en jij vindt dat andere gebruikers buiten je groep geen permissie moeten hebben om er zelfs maar in te kijken, zet je je `umask` op `007` en het resultaat wordt dan `750` of wel `rwxr-x---`.

Je manipuleert de `umask` vanuit je shell met

```
henk@boven:~/test> umask
0022
henk@boven:~/test>
```

Alleen `umask` zonder toevoegingen geeft de huidige `umask`. En `0022` betekent dat de `w` permissie van `g` en `o` weggemaskeerd worden. Zetten van `umask` is heel simpel:

```
henk@boven:~/test> touch hhh
henk@boven:~/test> l hhh
-rw-r--r-- 1 henk wij 0  9 jul 15:17 hhh
henk@boven:~/test> umask 027
henk@boven:~/test> touch jjj
henk@boven:~/test> l hhh jjj
-rw-r--r-- 1 henk wij 0  9 jul 15:17 hhh
-rw-r----- 1 henk wij 0  9 jul 15:18 jjj
henk@boven:~/test>
```

Dit zorgt er voor dat de `r` en `x` permissies voor `o` ook worden weggemaskeerd.

Als je altijd een andere `umask` dan de standaard in je shell wilt hebben (die dus ook geldt voor alles wat je vanuit die shell sessie start), kun je je `umask` commando het best in je `~/.profile` zetten.

Als laatste heb ik voor een bestand alle bovengenoemde bitjes aangezet. Dat slaat nergens op, maar ziet er leuk uit:

```
henk@boven:~/test> l hhh
-rwsrwsrwt 1 henk wij 0  9 jul 15:17 hhh*
henk@boven:~/test> stat hhh
  Bestand: ‘hhh’
  Grootte: 0            Blokken: 0            IO-blok: 4096   leeg normaal bestand
Apparaat: 806h/2054d   Inode: 6030810      Koppelingen: 1
Toegang: (7777/-rwsrwsrwt)   UID: (  500/    henk)   GID: (  500/     wij)
Toegang:   2013-07-09 15:17:36.810544578 +0200
Gewijzigd: 2013-07-09 15:17:36.810544578 +0200
Veranderd: 2013-07-09 16:29:04.017054304 +0200
Ontstaan:  -
henk@boven:~/test>
```

Je ziet hier dat de drie bits `s`/`S`, `s`/`S` en `t`/`T` als een extra octade vooraan worden getoond. Dat verklaart ook de extra `0` die daar vaak staat, bijv bij het tonen van `umask` hierboven.