GREP, AWK, SED a jiné filtry v LINUXu

Vybrané filtry

more *Program se zastaví vždy po vypsání jedné stránky textu. Ukončení Ctrl+C*

less *Umožňuje pohybovat se v souboru dopředu i dozadu.*

head [**-*n***] *Zobrazí začátek souboru, prvních 10 řádků nebo n řádků*

tail [**-*n***] *Zobrazí konec souboru, posledních 10 řádků nebo n řádků*

sort [**-n**] *Setřídí soubor podle abecedy, je-li zadáno –n dojde k setřídění podle numerických hodnot.*

uniq *Odstraní stejné řádky, pokud následují ihned za sebou.*

wc [**-l|-w|-c**] *Vypíše počet řádků (-l), slov (-w) a znaků (-c) v daném souboru.*

cut *Umožňuje zadat, kterou část řádky chceme zobrazit, může pracovat s poli nebo s jednotlivými znaky. Implicitně předpokládá, že jsou pole oddělena tabulátorem.*

cut -d *znak* *Explicitní určení oddělovacího znaku.*

cut -f *číslo* *Výběr určitého prvku pole podle parametru číslo.*

grep, fgrep egrep *Vybírají řádky, které obsahují nebo neobsahují zadaný prototyp řetězce znaků.*

awk *Textový procesor.*

sed *Proudový editor. Jedná se o řádkový editor, kterému se řídící příkazy zadávají z příkazové řádky a který čte opravovaný text buď ze souboru nebo ze standardního vstupu.*

Systémové programy jsou v Unixu koncipovány tak, aby plnily pouze jeden specializovaný úkol (například vyhledávání řetězců v textovém souboru). Tento úkol ale musí splnit spolehlivě a rychle. Řešení komplexního úkolu je proto nutné rozložit na jednoduché úkoly, pro jejichž řešení existují systémové programy. Protože unixové shelly umožňují vytváření rour a systémové programy mohou většinou pracovat jako filtry, je možné jednoduché sestavení specializovaných systémových programů do roury, která daný komplexní úkol vyřeší. Tento přístup k řešení úkolů je velmi flexibilní a efektivní. Autoři Unixu jej měli na mysli již od počátku vývoje systému a podpořili jej zabudováním systémové služby pro vytváření rour již do prvních verzí Unixu.

Regulární výrazy

Slouží pro zadávání prototypů řetězců v příkazech programů **grep**, **egrep**, **awk** a **sed**.

Regulární výraz obsahuje ***běžné znaky*** a ***znaky se speciálním významem*** (***metaznaky***).

Nejdůležitější **metaznaky** mají následující význam:

. *Označuje jeden libovolný znak*

\ *Jedná se o znak výluky, tj. ruší speciální význam bezprostředně následujícího znaku*

[ ] - ^ *Pomocí těchto znaků se vytváří výraz pro zadání množiny alternativních znaků (alternativy). Pravidla pro vytvoření alternativy jsou stejná jako v shellu při zadávání názvu souboru pomocí expansních znaků. Například*

`[a-z]` *označuje libovolný znak ze znaků* `a,...,z`

`[^adz]` *označuje jeden znak různý od znaků* `a,d,z`

`[.x]` *označuje znak* **.** *nebo* `x` .

^ (mimo hranaté závorky) O*značuje začátek řádku.*

`$` *Označuje konec řádku.*

`*` **(***uzávěr***)** *Uzávěr se vztahuje k bezprostředně předcházejícímu znaku a znamená libovolný počet opakování tohoto znaku. Libovolný počet zde znamená i nulový počet opakování.*

Příklad 1:

`ab*`	*označuje řetězce* `a` , `ab` , `abb` , `abbb` atd.
`.*`	*označuje libovolný řetězec* (i prázdný)
`..*`	*označuje libovolný neprázdný řetězec*
`\..*`	*označuje libovolný řetězec začínající znakem* **.**
`^$`	*označuje prázdný řetězec*
`[a-zA-Z]*`	*označuje libovolný alfabetický řetězec*

Programy grep, fgrep, egrep

Program **grep** slouží pro vyhledávání řádků, které obsahují nebo naopak neobsahují řetězec, který splňuje prototyp zadaný regulárním výrazem. Vstupní data **grep** čte ze souboru nebo ze standardního vstupu. Výstup programu je směrován na standardní výstup.

Základní syntaxe příkazu **grep** je

grep "*regulární výraz***"** [*soubor*]	*Na výstup propouští ty řádky, které splňují regulární výraz.*
grep -v "*regulární výraz***"** [*soubor*]	*Na výstup propouští ty řádky, které nesplňují regulární výraz.*

Příklad 2: Složený příkaz

```
$ls -al  /etc | grep "^d"
```

vypíše všechny podadresáře obsažené v adresáři `/etc` .

Program **fgrep** (*fast grep*) je rychlejší verze programu **grep**. Rychlejší může být proto, že tvar přípustných regulárních výrazů je podstatně omezen. Regulární výraz nesmí obsahovat žádný metaznak. Regulární výraz tedy degeneruje na řetězec normálních znaků a **fgrep** vyhledává v řádcích souboru zadaný řetězec znaků.

Příklad 3: Příkaz

```
$cat /etc/passwd | fgrep -v /bin/csh
```

vypíše řádky všech uživatelů, pro které je po připojení spuštěn jiný shell než `/bin/csh` .

Program **egrep** (*extended grep*) je rozšířená verse programu **grep** . Povoluje obecnější formu regulárních výrazů. Regulární výrazy lze především do sebe vnořovat.

Například regulární výraz lze zapsat takto:

`a(xy)*` označuje řetězce `a, axy, axyxy, axyxyxy` atd.

Kromě toho regulární výrazy mohou navíc obsahovat následující **metaznaky**

+	*Modifikace uzávěru *. Znamená jedno nebo více opakování znaku.*
?	*Uzávěr ve významu jedno nebo žádné opakování znaku.*
|	*Operátor, který má význam logické spojky nebo.*

Příklad 4: Výraz

`x+` je totéž co `[x]*`

Výraz

`(abc)?` splňuje prázný řetězec nebo řetězec `abc`

`xyz | ab` splňuje řetězec `xyz` nebo řetězec `ab`

Program sed

Program **sed** je ***proudový editor*** (***stream editor***, proto se užívá zkratka **sed**). Vznikl úpravou řádkového editoru **ed** a používá stejné instrukce jako editor **ed**. Na rozdíl od editoru **ed** čte příkazy z příkazového řádku a ne z klávesnice.

Editor **sed** se spouští takto

sed '*příkazy*' [*soubor* . . .]

sed čte vstupní data ze souborů uvedených v příkazové řádce. Pokud v příkazové řádce není žádný soubor uveden, načítá data ze standardního vstupu. Program **sed** načítá vstupní data

postupně po řádkách, řádky mění podle zadaných příkazů a pak je vypisuje na standardní výstup. Příkazy se oddělují středníkem. V Bourne shellu mohou být také odděleny znakem nový řádek.

Editor **sed** je především určen pro použití ve skriptech. Nejčastěji se používá tehdy, když je třeba určitý řetězec řádky zaměnit za jiný. Pro naše účely tento způsob použití postačí a proto se omezíme jen na ty příkazy, které jsou k tomu třeba. Zájemce o úplnou informaci o programu **sed** odkazujeme na odbornou literaturu (například Kernighan 1996).

Příkazy pro substituci jsou následující:

/ *vzor* / **s** /re*gulární_výraz* / řetězec /	*Ve všech řádcích, ve kterých je nalezena instance vzoru, je nahrazena první instance regulárního_výrazu řetězcem.*
/ *vzor* / **s** / re*guární_výraz* / řetězec / **g**	*Ve všech řádcích, ve kterých je nalezena instance vzoru, je nahrazena každá instance regulárního_výrazu řetězcem.*

Instancí regulárního výrazu rozumíme řetězec, který jej splňuje. Ve výše uvedených příkazech pro substituci jsou *vzor* a *regulární_výraz* regulárními výrazy. V obou příkazech pokud není uveden *vzor*, vezmou se všechny řádky.

Příklad 5:

```
$sed 's/ab/xy/'
ababab prvni
xyabab prvni
ctrl+d
$sed 's/[ab]/xyz/g
>s/prvni/druhy/'
ab prvni
xyzxyz druhy
ctrl+d
```

Poznámka

Program **sed** normálně opisuje na standardní výstup všechny řádky vstupního souboru. Pokud tento opis vypneme modifikátorem příkazu **-n** , vypíše jen ty řádky, pro které to explicitně požadujeme příkazem **p** .

Proto například příkaz

sed -n '/*vzor*/**p'** *soubor*

dá stejný výstup jako příkaz

grep "*vzor*" *soubor*

Program awk

Program **awk** je řádkově orientovaný program pro úpravu textu. Jedná se o program značné složitosti, jehož činnost lze řídit programem napsaným ve speciálním jazyce.

Při většině aplikací ale stačí řídit činnost **awk** pomocí sekvencí jednoduchých příkazů, které se zadají z příkazové řádky. Omezíme se pouze na tento případ jeho použití.

Program **awk** lze spustit následovně:

awk **'***příkazy***'** [*soubor*]

Pokud soubor není uveden, vstupní data se čtou ze standardního vstupu. Příkazy se oddělují středníkem nebo v Bourne shellu také znakem nový řádek.

Program **awk** načítá řádky ze souboru nebo ze standardního vstupu, modifikuje je podle zadaných příkazů a výsledek vypisuje na standardní výstup.

Načítané řádky program **awk** dělí na slova. Oddělovač slov je implicitně mezera. Pomocí modifikátoru **-F**, který se uvede v příkazové řádce, lze zadat i jiný oddělovač slov. Na slova se lze v příkazech, které řídí činnost **awk** , odvolávat pomocí symbolů **$1, $2, ...**

Nejpoužívanější syntaxe příkazů jsou následující :

podmínka

***podmínka* { *akce* }**

/ *regulární výraz* /

/ *regulární výraz* / { *akce* }

Podmínka se tvoří z *termínů* pomocí *relačních operátorů a logických operátorů* stejných jako v jazyce C a z operátoru ~ . Operátor ~ slouží pro srovnání regulárního výrazu s hodnotou obsaženou ve slově. *termíny* jsou buď celá čísla, regulární výrazy nebo slova. Regulární výrazy nutno uzavřít do závorek / /. Regulární výrazy v podmínkách se vztahují na slova, ne na řádky. Například regulární výraz `/^x/` splňují všechna slova začínající na `x` a regulární výraz `/^$/` označuje prázdné slovo.

Akce je v nejjednodušším případě přiřazení hodnoty slovu nebo příkaz pro tisk *print*

Řádky, které nesplňují zadanou *podmínku*, nebo ve kterých není nalezen řetězec splňující *regulární výraz* , se vynechají. S ostatními řádky se provede v příkazu specifikovaná akce. Většinou spočívá ve vytisknutí slov řádku zadaným způsobem. Pokud akce není uvedena, opíše se načtený řádek beze změny na výstup.

Příklad 6: Následující příkazy mají tento význam:

`$1 > 100`	*Pokud je první slovo větší než* `100` *vytiskne se celý řádek*
`/abc/{print $1,$3}`	*Pokud je v řádku nalezen řetězec* `abc`*, je vytištěno první a třetí slovo řádky*
`$3 ~ /^x/ && $5 == 10 {print "slovo1=",$1, "slovo6=",$6}`	*Pokud třetí slovo začíná na* `x` *a páté slovo je rovno* `10`*, vytiskne se první a šesté slovo řádku*

Příklad 7: Ze souboru uživatelů vypište uživatele, kteří mají *UID* větší než 100. Ve výpisu bude uživatelské jméno, *UID* uživatele a plné jméno uživatele (je v poli poznámky).

Řešení:

```
$awk -F :  '$3 > 100 {print  $1,$3,$5)'  /etc/passwd
```

Příklad 8: Ze souboru uživatelů vypište všechny uživatele, kteří mají do systému přístup bez uvedení hesla.

Řešení:

```
$awk -F : '$2 == "" '  /etc/passwd
```

nebo

```
$awk -F : '$2 ~  /^$/'  /etc/passwd
```

Vkládání dat do skriptu

Programu, který čte vstupní data ze standardního vstupu a který je spuštěn ze skriptu, lze vložit vstupní data přímo do scriptu jako tzv. ***here-document*** .

Provede se to tak, že se ve scriptu za příkaz spuštění programu uvede *řetězec*, který bude sloužit jako indikátor konce vstupních dat. Před tento *řetězec* nutno zapsat znaky << . Vstupní data se vloží počínaje řádkem, který následuje za příkazem pro spuštění programu.

Po spuštění začne program načítat vstupní data ze skriptu. Data se načítají dokud se nenarazí na zápis *řetězce* ukončujícího data. *Řetězec* indikující konec vstupních dat musí být uveden na samostatném řádku od první pozice.

Vložení here-dokumentu do skriptu:

. . . . . .
příkaz parametry << řetězec
vstupní data (here-document)
řetězec
. . . . . .

Příklad 9: následující skript `dopis` pošle všem uživatelům, kteří jsou uvedeni v souboru `uzivatele`, e-mail, jehož text je vložen přímo do skriptu.

```
$cat dopis
mail `cat uzivatele` <<+
pripominam stredecni schuzku
                    novak
+
```

Příklady

Příklad 10: Vypište prvních sedm a posledních pět řádků souboru `/etc/passwd`.

Řešení

```
$cat /etc/passwd | head -7
$cat /etc/passwd | tail -5
```

Příklad 11: Vyzkoušejte použití filtrů `sort` a `uniq`.

Řešení

```
$cat > pokus1
99 balonku
101 dalmatinu
16 svicek
24 hodin
9 studentu

$cat pokus1 | sort
101 dalmatinu
16 svicek
24 hodin
99 balonku
9 studentu

$cat pokus1 | sort –n
9 studentu
16 svicek
24 hodin
99 balonku
101 dalmatinu
```

```
$cat > pokus2
jedna
dva
jedna
jedna
dva
dva
tri
tri
jedna

$cat pokus2 | uniq
jedna
dva
jedna
dva
tri
jedna

$cat pokus2 | sort | uniq
dva
jedna
tri
```

Příklad 12: Vypište počet znaků, slov a řádků v souboru `/etc/passwd`.

Řešení

```
$cat /etc/passwd | wc
   6   16   85
$cat /etc/passwd | wc -c
   85
$cat /etc/passwd | wc -w
   16
$cat /etc/passwd | wc -l
   6
```

Příklad 13: Zjistěte název interpretu příkazů uživatele `john`.

Řešení

```
$cat /etc/passwd | grep ^john: | cut –d : -f 7
```

Poznámka: *Název interpretu je 7.položka, položky jsou odděleny dvojtečkou.*

Příklad 14: Přepište obsah souboru `/etc/passwd` do vašeho domácího adresáře pod názvem hesla . Při tomto přepisu zaměňte všechny výskyty znaku **:** za znak **?** .

Řešení

```
$sed 's/:/?/g' >~/hesla
```

Příklad 15: Napište příkaz, který vypíše z adresáře `/etc` pouze podadresáře.

Řešení

```
$ls -al /etc |grep '^d'
```

Příklad 16: Napište příkaz, který vypíše počet podadresářů adresáře `/etc` .

Řešení

```
$ls -al /etc |grep '^d' |wc -l
```

Příklad 17: Vytvořte dlouhý výpis adresáře `/etc` (příkazem **ls -al**). Dlouhý výpis bude obsahovat pouze adresáře a začátek každého řádku bude místo písmenem `d` začínat řetězcem `Adresar:`

Řešení

```
$ls -al /etc |grep '^d' |sed 's/^d/Adresar:'
```

Nebo lepší řešení:

```
$ls -al /etc |sed -n '/^d/s/^d/Adresar:/p'
```

Příklad 18: Vypište počet uživatelů, kteří mají v systému momentálně spuštěny nějaké procesy. Pro výpis procesů využijte příkaz `ps –ef`.

Řešení

```
$ps –ef | awk '{print $1}' | sort | uniq | wc -l
```

Příklad 19: Pomocí příkazu **awk** vypište z tabulky uživatelů `/etc/passwd` jména těch uživatelů, kteří mají *UID* větší naž 100.

Řešení

```
$cat /etc/passwd | awk -F : '$3 =>100  {print $1 }'
```

Příklad 20: Příkazem vypište

a) Seznam připojených uživatelů

b) Seznam studentských uživatelských jmen (tj. uživatelských jmen začínajících na `x`)

c) Seznam uživatelů, kteří zabírají na disku více než 1MB

Řešení

Bourne shell:

a)

```
$who | awk '{print $1}'
```

b)

```
$cat /etc/passwd | awk -F : '$1 ~ /x.*/ {print $1}'
```

c)

```
$du -sk /home/* 2>du.err |awk '$1 > 1024 {print $2}' | \
     awk -F / '{print $3}'
```

V C shellu se příkazy a) a b) zadají stejně. Příkaz c) lze zadat následovně:

```
%(du -sk /home/* | awk '$1 > 1024 {print $2}' | \
     awk -F / '{print $3}' >/dev/tty) >&/dev/null
```

Příklad 21: Napište příkaz, který zjistí kolik je v souborech `/etc/passwd` a `/etc/group` prázdných řádků.

Řešení

```
%grep "^$" /etc/passwd /etc/group | wc -l
```