Institut für Informatik
Praktische Informatik und Bioinformatik
Prof. Dr. Ralf Zimmer

Wintersemester 2018/19

Übungen zum Bioinformatik-Tutorium

Blatt 2

Termin: Dienstag, 30.10.2018, 11 Uhr

1. Wildcards und Brace Expansion

(a) Finde heraus wie man mit `mkdir` einen Pfad mit seinen Sub-Ordnern erstellen kann, und erstelle dann mit Hilfe von Brace Expansion in einem Befehl das Verzeichnis `wildcards/ordnerXX` wobei `XX` für die Zahlen von 01 bis 10 steht.

`mkdir -p wildcards/ordner{01..10}`
`-p` lässt `mkdir` ineinander liegende Ordner in einem Befehl erzeugen. Es wird also der noch nicht existierende Ordner `wildcards` *und* die darin liegenden `ordnerXX` auf einmal erzeugt. Es werden also die Elternordner (parents) miterzeugt.
`ordner{01..10}` ist Brace Expansion wie auf den Folien: Es sollten eigentlich Ordner die `ordner01`, `ordner02`, ... `ordner10` entstehen. Bei Mac-Usern kam es vor, dass die führende null ignoriert wird, also `ordner1`, ... erzeugt wird.

(b) Erstelle nun in `ordner01` 1000 Dateien mit den Dateinamen `0001, 0002, ..., 1000`.

`touch {0001..1000}` (natürlich vorher in `ordner01` wechseln!)
Zur Information: Falls ihr diese Dateien löschen wollt, und ihr bei einem Versuch auf der Commandline wegen jedem einzelnen File gefragt werdet ob ihr euch auch wirklich sicher seid: Löscht es einfach über den grafischen Dateimanager, denn die Alternative es rekursiv zu forcieren mit `rm -rf *` wäre *sehr, sehr leichtsinnig*! Warum? Wenn ihr ausversehen ein, zwei Ordnerebenen darüber seid, wird stattdessen von dort aus rekursiv `*` gelöscht, also: *Alles.*

(c) Versuche dir nun mit `echo` und Wildcards nur folgende Dateinamen ausgeben zu lassen:

(i) Nur solche, die irgendwo im Namen die Zeichenfolge 20 enthalten.

(ii) Nur solche die an den ersten 3 Stellen eine 0 haben

(iii) Alle, die *nicht* die 1 an erster Stelle haben

Die Lösungen zu dieser Übung sind *Vorschläge*, es gibt viele andere Wege um diese Aufgaben zu lösen.

(i) `*20*`
Man könnte meinen, dass diese Wildcard nicht passt da `*20*` aussieht als müsste vor und hinter der `20` ein Zeichen kommen; also dass eine `20` am Anfang/Ende einer Zeichenkette nicht erkannt wird. *Aber:* `*` kann auch einen 0-langen String matchen!
(ii) `echo 000?`
`000` wird explizit hingeschrieben, da wir wissen dass zu Beginn genau dieses Muster vorkommen soll. `?` am Ende würde zwar auch nichtnumerische Zeichen matchen, was uns aber in diesem Fall nicht stört.
(iii) `echo [!1]*`
Hier gab es das Problem, dass auf den CIP-Rechner standardmäßig eine Shell namens `zshell` eingerichtet ist, die diese Negationssyntax (`[!1]`) nicht unterstützt. Standardmäßig arbeiten die meisten Linux-Distributionen allerdings mit der `bash`.
In `bash` würde `[!1]*` bedeuten: "hier keine 1 (`[!1]`), der Rest ist egal (`*`)"

2. Pipes und Umleitungen

(a) Finde heraus, wie man mit `tail` eine Datei auf Änderungen beobachtet.

`tail -f datei` (steht recht weit oben in der manpage)
Häufig kam die Frage auf, wozu das nutzt: Wie in den Folien zu den Umleitungen bereits erwähnt wird häufig an das Ende eines Files etwas hinzugefügt (konkateniert); dieses Kommando erleichtert sozusagen die Überwachung dessen was laufend in die Datei geschrieben wird.

(b) Die Datei `~/tutorium/material/yeastract.csv` enthält das genregulatorische Netzwerk von Hefe. In der ersten Spalte steht ein Transkriptionsfaktor und in der zweiten Spalte ein Gen dessen Expression durch den Transkriptionsfaktor reguliert wird. Suche nun mit `grep` nach jedem Eintrag, in dem das Gen mit dem Identifier **YDR213W** erwähnt wird und zähle Diese.

```
grep YDR213W yeastract.csv | wc -l
```
Es sollten 217 Zeilen sein. `grep` gibt hier nur die Zeilen aus die den String `YDR213W` enthalten, welche an `wc -l` weitergeleitet werden, wodurch nur Zeilen mit dem gesuchten Inhalt gezählt werden.

(c) Finde nun alle Gene die durch den Transkriptionsfaktor **YGL073W** reguliert werden. Speichere die Identifier der regulierten Gene (ohne Duplikate) in eine neue Datei in deinem Tutoriumsordner. Schaffst du das mit einem Befehl? (Zur Kontrolle: Es sollten 571 einzigartige Gene sein.)

```
grep '^YGL073W' yeastract.csv | cut -d ',' -f 2 | sort | uniq >
~/tutorium/genefile
```
Wenn eine falsche Anzahl herauskommt, liegt es vermutlich daran, dass *beide* Spalten gezählt wurden. In der Aufgabenstellung wird nur nach den von `YGL073W` regulierten Genen (also 2. Spalte) gefragt.

Dazu muss man nur die `YGL073W` mit `grep` ausgeben die in der ersten Spalte sind: entweder mit `^YGL073W` nur die Identifier auswählen die am Zeilenanfang stehen, oder mit `YGL073W,` nur die auswählen die links von einem Komma stehen.
`cut` lässt mit `-d` den `d`elimiter auswählen, also an welchem Trennsymbol geschnitten werden soll. `-f` lässt einen die Spalte(n) auswählen die man ausgegeben haben möchte.
`sort` und `uniq` eliminieren dann sämtliche Duplikate, und mit `>` wird dann der letztendliche Output in den angegebenen File geschrieben anstatt auf stdout.

(d) Die Datei `~/tutorium/material/cathscop.inpairs` enthält in den ersten beiden Spalten Paare von Protein IDs. In den nächsten beiden Spalten stehen die jeweils zugehörigen CATH-Nummern. Wie viele einzigartige CATH-Nummern enthält diese Datei?

```
cut -d ' ' -f 3 cathscop.inpairs > col3
cut -d ' ' -f 4 cathscop.inpairs > col4
cat col3 col4 | sort | uniq | wc -l
```

Es sollten 901 einzigartige CATH-Nummern vorkommen.
Läuft analog zu den vorigen Aufgaben, mit dem Unterschied dass hier die Spalten einzeln in Files gespeichert werden, und dann mit `cat` konkateniert ausgegeben werden.

(e) Die Datei `~/tutorium/material/C_elegans.pep.all.fasta` enthält das Referenzproteom von C. elegans im FASTA Format. Speichere alle FASTA header in eine neue Datei. Wie viele Proteine sind in dieser Datei? Sortiere die Namen der Proteine alphabetisch und gib die ersten 20 aus.

```
grep '>' C\_elegans.pep.all.fasta > \textapprox/tutorium/worm_proteins
wc -l worm_proteins
cut -f 1 -d ' ' worm_proteins | sort | head -n 20
```

Um herauszufinden wie die Header markiert sind (>) hätte man entweder Suchen, Fragen oder den File aufmerksam ansehen müssen.

3. Packen und Entpacken

(a) Entpacke und öffne die Datei `/usr/share/doc/bash/README.commands.gz`.

Da ihr keine Berechtigungen habt um es in dem eigentlichen Ordner zu entpacken, kopiert ihr es erst mal zu euch rüber in euren Ordner (oder wo auch immer ihr es haben wollt):
`cp /usr/share/doc/bash/README.commands.gz ~/tutorium/`
Dann mit `gzip` entpacken (siehe Dateiendung - falls die Dateiendung fehlt kann man mit dem Befehl `file` oft herausfinden worum es sich handelt):
`gzip -d README.commands.gz`

(b) Packe das Verzeichnis `/usr/share/doc/bash/` als `.tar.gz`.

```
tar -cf archiv /usr/share/doc/bash/
```

4. Arbeiten mit der Shell

Informiere dich auf `http://www.ensembl.org/info/website/upload/gff.html` über das `gtf`-Format und Betrachte im Folgenden die Datei
`~/tutorium/material/saccharomyces_cerevisiae/Saccharomyces_cerevisiae.R64-1-1.75.gtf`
Benutze shell-Kommandos um folgende Fragen zu beantworten. Hinweis: Das Anlegen eines symlinks von der `gtf`-Datei irgendwo in deinen tutoriums-Ordner hinein erspart einiges an Tipparbeit.

(a) Speichere die Zeilen, die eine Coding Sequence (CDS) definieren in eine Datei. Kontrolliere, dass es 7055 sind und korrigiere gegebenenfalls Fehler.

Der Schritt mit dem symlink ist optional, erspart einem aber frenetisches Hämmern auf Tab:

```
ln -s (pfad zum ordner)/Saccharomyces_cerevisiae.R64-1-1.75.gtf yeast.gtf
grep 'CDS' yeast.gtf > cds
wc -l cds
```

(b) Finde heraus welche verschiedenen Features in der Datei gespeichert sind.

```
grep -v '#' yeast.gtf | cut -f 3 | sort | uniq
```

Da `#` das Symbol ist um ein Kommentar in `gtf`-Files zu markieren, lassen wir mit `-v` nur solche Zeilen ausgeben, die es *nicht* enthalten.

(c) Speichere alle Pseudogene auf dem "+" Strang in eine Datei. Wie viele Pseudogene gibt es auf dem "–" Strang?

```
grep pseudogene yeast.gtf | grep -w '+' > pseudoplus
grep pseudogene yeast.gtf | grep -w '-' | wc -l
```

Hier nutzen wir `-w` um zu vermeiden, dass `+` bzw. `-` in einem Wort gematcht wird, da wir nur ein "freistehendes" Symbol wie in der entsprechenden Spalte matchen wollen.