Aufgabe 4

Shell-Skripte, reguläre Ausdrücke

Aufgabenstellung

In dieser Aufgabe sollt ihr ein Shell-Skript schreiben, welches bestimmte Informationen aus den **FAQ**, der **Gruppeneinteilung** sowie dem **Terminkalender** der Unix-Übung extrahiert.

Das Programm wird entsprechend der folgenden Usage aufgerufen:

```
Usage:
finder.sh -h | finder.sh --help
  print this help and exit

finder.sh HTML-FILE OPTION

  where HTML-FILE
    is a file ending with .html and beginning with a specific PREFIX for each OPTION

  where OPTION is one of
    -s CHARS, --search CHARS
         extract and print all <h3>-headlines containing the given character string
         where CHARS
               is the searched for character string
         and PREFIX for HTML-FILE is: faq

    -g SID, -g GROUPNO, --group SID, --group GROUPNO
         extract and print the timeslot, group and student-ID(s) for a given group number or student-ID
         where SID
               is a string containing 3 or 4 alphabetic signs (optional) followed by exactly 6 digits
         where GROUPNO
               is a two-digit number
         and PREFIX for HTML-FILE is: gruppen

    -c DAY SUBGROUP, --calendar DAY SUBGROUP
          extract all calendar dates belonging to the given weekday and subgroup
          where DAY
                is a weekday between 'Montag' and 'Freitag' (in German)
          where SUBGROUP
                is either A or B
         and PREFIX for HTML-FILE is: kalender
```

Aufruf

Im einfachsten Fall ruft man das Programm mit dem Parameter `-h` oder `--help` auf, wodurch es den oben angegebenen Usagetext als Hilfe ausgibt.

Eine Angabe von weiteren Parametern bei der Hilfeanfrage ist nicht erlaubt und somit als Fehlerfall zu werten.

Sollte nicht die Hilfe aufgerufen werden, kann das Programm eine der folgenden Analysen vornehmen:

- Eine Schlagwortsuche auf den entsprechenden Überschriften eines FAQs (entsprechend des FAQs zu den Übungen).
- Die Ausgabe von Gruppeninformationen anhand der Gruppen- oder einer Matrikelnummer auf Basis einer Tabelle, die der Tabelle für die Gruppeneinteilung in der Unix-Übung entspricht.
- Eine Terminsuche über die Angabe von Wochentag und Teilgruppe (A oder B) in einem tabellarischen Kalender, der dem Format des Kalenders der Unix-Übung entspricht.

Verarbeitung

Im Erfolgsfall wird die Ausgabe eures Programms entsprechend der Spezifikationen der einzelnen Optionen auf `stdout` geschrieben. Als Zeilenumbruch wird hierbei stets `\n` verwendet.

Eine Überprüfung des Dateinamens der HTML-Datei findet statt:
Gültige Eingabe-Dateien müssen stets die Dateiendung “html” haben und entsprechend der gewählten Option über ein spezielles Präfix verfügen (siehe auch Hinweis bei “Fehlerbehandlung”). Wir gehen der Einfachheit halber davon aus, dass die Dateinamen immer komplett klein geschrieben sind.

FAQ-Suche

Die Schlagwort-Suche sucht in der übergebenen HTML-Datei in allen `<h3>`-Überschriften nach der übergebenen Zeichenkette. Die Groß- und Kleinschreibung ist hierbei nicht relevant. Die Eingabe von z.B. “serverTEST” führt also zum gleichen Ergebnis wie die Eingabe von “Servertest”.

Als Ergebnis werden die Überschriften, also alles was in dem jeweiligen `h3`-Tag steht, die die gesuchte Zeichenkette enthalten, zeilenweise auf `stdout` ausgegeben. Jede Zeile enthält dabei genau eine Überschrift.

Beispiel:

Die folgende Anfrage:

```
./finder.sh cache/faq.html -s kom
```

liefert diese Ausgabe:

```
Was sind die Kommunikationskanäle für die Übungen?
Ich habe ein Problem oder eine Frage, welche zu komplex ist um sie textuell als Ticket zu beschreiben. Wo bekomme ich einen Besprechungst
Wie kommuniziert ihr mit uns?
Ich verstehe etwas an der Aufgabenstellung / dem vorgegebenen Code (falls vorhanden) / den vorgegebenen Kommentaren (falls vorhanden) nic
Ich habe weniger als 100% im Servertest und beim besten Willen keine Idee woran es liegen könnte. Wo bekomme ich Hilfe?
Ich hab das FAQ komplett gelesen und jetzt traue ich mich nicht mehr euch was zu fragen.
```

Wenn keine passenden Einträge gefunden werden, so wird auch nichts ausgegeben.

Hinweis: Die gesuchte Zeichenkette kann sowohl mehrere Worte als auch nur einen Wortteil umfassen. Das Ergebnis der Suche nach “nikation” z.B. ist also eine (ggfs. unechte) Obermenge des Ergebnisses der Suche nach “Kommunikation”, was wiederum eine (ggfs. unechte) Obermenge des Ergebnisses der Suche nach “Welche Kommunikation” ist.

Um abzusichern, dass nur in einer strukturell passenden HTML-Datei gesucht wird, wird bei der übergebenen HTML-Datei das Präfix “faq” im Dateinamen erwartet.

Kalender-Ausgabe

Die Kalender-Ausgabe ermöglicht es, anhand der Angabe eines Wochentags (Montag, Dienstag, Mittwoch, Donnerstag oder Freitag) sowie der Angabe einer Teilgruppe (also `A` oder `B`) alle regulären Abnahmetermine aus dem Kalender zu extrahieren, die für diese Auswahl vorgesehen sind. Die Testabnahmen werden hierbei nicht mit ausgegeben.

Eine korrekte Eingabe gemäß dieser Anforderung muss sichergestellt sein, allerdings gilt hierbei wieder für beide Parameter, dass Groß- und Kleinschreibung nicht relevant ist.

Die Ausgabe erfolgt in folgendem Format:

- In der ersten Zeile werden die Informationen zur gewählten Teilgruppe und dem gewählten Wochentag ausgegeben. Hierbei erfolgt die Ausgabe der Gruppenangabe stets in Großbuchstaben und die der Wochentags-Beschreibungen als Adverbien. Die Ausgabesyntax sieht (in pseudo-regulärer Notation) wie folgt aus: `Gruppe (A|B) - (montags|dienstags|mittwochs|donnerstags|freitags)`
- Danach werden die gefunden Termine zeilenweise als “Baumstruktur” in Reihenfolge ihres Auftretens in der Datei ausgegeben. Hierbei wird jeder gefundene Termin mit einem Präfix von “`|-`” (Pipe Minus Leerzeichen) versehen. Weitere Leerzeichen dürfen nicht vorhanden sein.

Beispielaufruf:

```
./finder.sh cache/kalender.html -c DiensTag a
```

Ausgabe:

```
Gruppe A - dienstags
|- 19.05.
|- 02.06.
|- 16.06.
|- 30.06.
|- 14.07.
```

Wenn keine passenden Einträge gefunden werden, so wird lediglich die erste Zeile ausgegeben.

Um abzusichern, dass nur in einer strukturell passenden HTML-Datei gesucht wird, wird bei der übergebenen HTML-Datei das Präfix “kalender” im Dateinamen erwartet.

Tipp: Schaut euch den Kalender nicht nur als HTML-Datei, sondern auch im Browser an, um herauszufinden, wie ihr bestmöglich an die gesuchten Infos kommen könnt. Es kann z.B. durchaus nützlich sein die einzelnen Wochentage in ein programmatisch besser verwertbares Format umzusetzen. Nicht jeder Teilaspekt der Aufgabe ist notwendigerweise durch die Anwendung von regulären Ausdrücken am geschicktesten lösbar.

Gruppen-Ausgabe

Die Gruppen-Ausgabe ermöglicht es, anhand von entweder einer Gruppennummer oder einer Matrikelnummer (mit oder ohne Studiengangsbezeichner) aus einer Gruppeneinteilungstabelle Informationen über eine Gruppe, ihre Mitglieder und ihren Abnahmetermin abzufragen.

Hierfür muss entweder eine genau zweistellige Gruppennummer angegeben werden, oder eine Matrikelnummer. Die Matrikelnummer muss hierbei mindestens aus genau 6 Ziffern bestehen und kann optional ein Präfix aus 3 oder 4 beliebigen Buchstaben haben. Eine Beschränkung auf die exakten Studienrichtungen ist hierbei nicht notwendig bzw. sinnvoll, da sie wenig Mehrwert hat, dafür aber eine Erweiterung um weitere Studiengänge immer Änderungen im Code erfordern würde. Die Studienrichtung vor der Matrikelnummer unterscheidet analog zu den anderen Eingaben auch nicht zwischen Groß- und Kleinschreibung.

Die Ausgabe erfolgt stets im folgenden Format:

- In der ersten Zeile erfolgt die Angabe der Gruppe im Format:
  `Gruppe:`, gefolgt von 7 Leerzeichen, gefolgt von der Gruppennummer.
- In der zweiten Zeile stehen ihre Mitglieder im Format:
  `Mitglied(er):`, gefolgt von einem Leerzeichen, gefolgt von der/den Matrikelnummer/n von entweder einem Gruppenmitglied (falls nur eins vorhanden ist) oder beiden Gruppenmitgliedern (falls zwei vorhanden sind), dann getrennt durch genau ein Leerzeichen.
- In der dritten Zeile erfolgt die Angabe des Abnahmetermins im Format:
  `Termin:`, gefolgt von 7 Leerzeichen, gefolgt von Angabe des Abnahmetermins genau so wie er im Kalender steht.

Weitere Leerzeichen außer den angegebenen sind in der Ausgabe unzulässig.

Beispiel:

Für eine Eingabedatei, die unter anderem die folgenden Zeilen enthält:

```
<table>
<thead>
<tr>
<th style="text-align:center;">Termin</th>
<th style="text-align:center;">Gruppe</th>
<th colspan="2" style="text-align:center;">Mitglieder</th>
```

```
</tr>
</thead>
<tbody>
<tr>
<td style="text-align:center;">Dienstag 15:30–16:45 (WInf: Gruppe A)</td>
<td style="text-align:center;">06</td>
<td style="text-align:center;">winf012345</td>
<td style="text-align:center;">winf543210</td>
</tr>
<tr>
<td style="text-align:center;">Donnerstag 14:00–15:15 (Inf: Gruppe B)</td>
<td style="text-align:center;">12</td>
<td style="text-align:center;">inf000000</td>
<td style="text-align:center;"></td>
</tr>
</tbody>
</table>
```

liefern die Anfragen:

```
./finder.sh cache/gruppen_minimal.html -g 012345
./finder.sh cache/gruppen_minimal.html -g winf543210
./finder.sh cache/gruppen_minimal.html -g 06
```

das folgende Ergebnis:

```
Gruppe:       06
Mitglied(er): winf012345 winf543210
Termin:       Dienstag 15:30–16:45 (WInf: Gruppe A)
```

und die Anfrage:

```
./finder.sh cache/gruppen_minimal.html -g 12
```

das folgende Ergebnis:

```
Gruppe:       12
Mitglied(er): inf000000
Termin:       Donnerstag 14:00–15:15 (Inf: Gruppe B)
```

Wenn die gesuchte Eingabe (Matrikelnummer oder Gruppe) nicht gefunden werden kann, so wird auch nichts ausgegeben.

Um abzusichern, dass nur in einer strukturell passenden HTML-Datei gesucht wird, wird ein bei der übergebenen HTML-Datei das Präfix “gruppen” im Dateinamen erwartet.

Parsen und Besonderheiten

Jede der drei Seiten hat ein anderes Format in Bezug auf den Aufbau. Analysiert daher die im **Archiv** enthaltenen Webseiten genau, bevor ihr mit dem Parsen beginnt.

Ihr könnt euch darauf verlassen, dass die relevanten HTML-Strukturen der Seiten immer erhalten bleiben - eine Gruppeneinteilungsseite wird also z.B. immer eine Tabelle mit vier Spalten beinhalten, in der die relevanten Informationen im vorgegebenen Format stehen. Whitespaces, Zeilenumbrüche und Einrückung können hierbei jedoch - wie in allen HTML-Dateien - beliebig sein. Jeglicher zusammenhängender Whitespace entspricht in HTML einem Leerzeichen. Beachtet zudem, dass in HTML-Dateien alle Arten von Zeilenumbrüchen auftreten können (`\n`, `\r`, oder `\r\n`). Zudem können HTML-Tags selbst in beliebiger Groß- und Kleinschreibung notiert sein und optional Attribute enthalten. HTML Dateien können außerdem Kommentare enthalten, diese entsprechen dann dem folgenden Format: Sie beginnen stets mit der Zeichenfolge `<!--` und enden mit der Zeichenfolge `-->`. Auskommentierte Inhalte sollen bei der Verarbeitung nicht berücksichtigt werden.

Zur Verdeutlichung der möglichen Bandbreite an Formatierungen haben wir bereits diverse unterschiedlich formatierte Varianten der Eingabe-Dateien im Archiv vorgegeben (siehe auch “Wichtiger Hinweis zum Servertest”).
Es bietet sich zur Analyse der Dateien an, aus der Vorlesung bekannte Tools wie z.B. `diff` einzusetzen, um die Unterschiede schneller zu erkennen.

Selbstverständlich soll eure Implementierung möglichst robust sein, also auf Veränderungen außerhalb der für die Ausgabe relevanten Bereiche tolerant reagieren. Es bietet sich dafür an, die für die Ausgabe relevanten Bereiche zunächst anhand von HTML-Tags oder HTML-Tag-Kombination zu identifizieren und den Rest der Seite einfach zu ignorieren.

Fehlerbehandlung

Im Fehlerfall soll wie üblich eine aussagekräftige einzeilige Fehlermeldung mit dem Präfix `ERROR:` gefolgt von der Usage auf `stderr` ausgegeben werden. Passend dazu soll der Exitcode des Programms im Fehlerfall selbstverständlich != 0 sein.

Die Aufrufsyntax des Programmes ist recht einfach gehalten, sie muss natürlich dennoch wie üblich auf Korrektheit geprüft werden (Anzahl Parameter etc.).
Zudem müssen alle Eingabeparameter gemäß ihrer Definition auf Korrektheit geprüft werden.

Zur Info: Die Prüfung des Dateinamens der angegebenen HTML-Datei soll hierbei verhindern, dass durch die Angabe von unpassenden Dateien (bezüglich der Inhalte) Verwirrung entsteht - eine vollständige Validierung der Eingabedateien auf ein gültiges und sinnvolles Format würde den Rahmen einer Aufgabe völlig sprengen.

Modularisierung

Nutzt auch in dieser Aufgabe wieder Funktionen um euer Programm *sinnvoll* zu modularisieren. Alle Hinweise aus **Aufgabe 3** gelten auch diesmal wieder.

Nutzung von Schleifen

Die Nutzung von Schleifen ist in dieser Aufgabe weder sinnvoll noch erlaubt.

Effizienz

Achtet wie immer bei eurer Lösung auf eine gewisse Effizienz. Dies heißt auch, dass ihr die Datei nicht mehrfach einlesen dürft, sondern deren Inhalte bei Bedarf in eurem Skript zwischenspeichern müsst.

Tipps zur Verarbeitung von HTML-Dateien

Bei der Verarbeitung von HTML-Dateien bietet sich stets eine Vorverarbeitung der Eingaben an, die diese in ein Format bringt, mit dem ihr besser arbeiten könnt. Es ist zudem nicht notwendig, während der Verarbeitung ein korrektes HTML-Format beizubehalten. Wichtig ist nur, dass die relevanten Inhalte während der Verarbeitung “nicht beschädigt” werden. So ist es zum Beispiel häufig sinnvoll, komplette Tags durch ein bestimmtes Zeichen zu ersetzen, um danach weniger komplizierte reguläre Ausdrücke zu haben - dabei sollte man allerdings darauf achten, nur Zeichen zu verwenden, die dann im entsprechenden Kontext eindeutig sind.

Umfang der Lösung

Diese Aufgabe ist mit weniger als 150 Zeilen Code lösbar (ohne Kommentare und Leerzeilen). Lösungen, welche mehr als 250 Zeilen (ohne Kommentare und Leerzeilen) umfassen, werden wir daher nicht akzeptieren.

Sollte eure Lösung also entsprechend umfangreich sein, überprüft diese auf weitere Vereinfachungsmöglichkeiten.

Sonstige Anforderungen

Eine sinnvolle Formatierung, Einrückung des Quelltextes und Kommentierung und eigenes Testen wird selbstverständlich vorausgesetzt.

Alle Hinweise aus **Aufgabe 3** gelten auch diesmal wieder.

Benennung

Nennt euer Skript “`finder.sh`”, damit die automatischen Tests laufen und achtet darauf, dass es auch als ausführbare Datei eingecheckt ist.

Erlaubte Tools

In dieser Aufgabe sind Tools wie etwa `perl`, `awk` und `ruby` nach wie vor nicht erlaubt und zur Lösung auch nicht vonnöten. Ebenso ist die Verwendung von Schleifen ausgeschlossen.

Dafür ist in dieser Aufgabe die Nutzung der `bash` ausdrücklich erlaubt, sofern ihr in der Abnahme erläutern könnt, welche Vorteile ihr davon habt und nutzt. Die Aufgabe lässt sich aber ebenso problemlos mit der `dash` lösen - ihr müsst keine `bash` verwenden.

Testen

Auch bei dieser Aufgabe werden wir eure Lösungen automatisiert mit dem Testserver überprüfen. Ihr könnt diesen auch nach wie vor selbst nutzen um sicherzustellen, dass ihr alle Anforderungen richtig gelesen und umgesetzt habt.

Es gelten nach wie vor alle Hinweise aus **Aufgabe 3** zum Testen.

Zum Entwickeln der regulären Ausdrücke bietet es sich an diese direkt in der Konsole zu testen und weiterzuentwickeln, bevor sie in das Programm integriert werden. Dies spart eine Menge Zeit und verhindert ständiges Hin- und Herspringen zwischen Editor und Konsole.

Wichtiger Hinweis zum Servertest

Wir werden für unsere Funktions-Tests nur die mitgelieferten HTML-Dateien verwenden. Sofern euer Programm diese alle fehlerfrei verarbeiten kann, müsst ihr euch also keine Sorgen um unerwartete Überraschungen in den HTML-Dateien machen.

Die Dateien sind stets in Gruppen eingeteilt. Es gilt hierbei, dass Dateien mit dem selben Präfix bei gleichlautenden Aufrufen auch die gleichen Ergebnisse liefern. Einzige Ausnahme ist die Datei gruppen_minimal.html, die für den Beispieltest eine anonymisierte und verkürzte Gruppentabelle enthält.

Nicht-funktionale Mängel

Nachbesserungsfähige nicht-funktionale Mängel:

Nachbesserungsgrund	Abzug Prozentpunkte bei Nichterfüllung
Repository / Ordnerstruktur schlecht gepflegt	5
Interpreterangabe fehlt	5
Uneinheitliche Formatierung oder Sprachwahl für Bezeichner oder Kommentierung	5
Unangemessene Bezeichner (z.B. hinsichtlich ihrer Länge oder Aussagekraft)	5
Mangelnde Codehygiene (z.B. toter Code)	5
Mangelhafte Formatierung / schlecht lesbarer Code (z.B. Einrückung, Tabs und Leerzeichen gemischt)	10
Mangelhafte Dokumentation (z.B. Funktionsköpfe / Inline-Kommentierung)	10
Unnötige Codeverdopplung	10
Verwendung von unstrukturierter Programmierung (z.B. unangemessen viele returns oder exits)	10
Verwendung von globalen Variablen in Funktionen	10

Nachbesserungsgrund	Abzug Prozentpunkte bei Nichterfüllung
Mangelnde Effizienz / umständliche Lösung (z.B. unnötige temporäre Dateien / Variablen)	20
Nicht vorhandene Testfälle	20
Umständliche / zu lange Lösung	30
Nicht aufgabenkonforme Implementierung (z.B. Schleifen außerhalb der Parameterverarbeitung)	30

Downloads

Den Beispieltest findet ihr zusammen mit den Cache-Dateien im **Archiv** zur Aufgabe.

Da ihr bereits mit Arnold und allen weiteren üblicherweise nicht in den Aufgabenstellungen spezifizierten Konzepten vertraut seid und die Aufgabe in der HTML-Aufgabenstellung bereits ausreichend erläutert ist, gibt es diesmal keine Folien.

Viel Erfolg!

letzte Änderung: 20.06.2020 19:47