makefile

décrire le graph de dépendances sous la forme de règles

Cible (target)

état à atteindre

- fichier à construire
- phony: “fichier virtuel”

Dépendance (dependency)

fichier nécessaire à la construction ou l'utilisation de la cible

Commandes (command)

commandes externes (`/bin/sh` par defaut) à executer pour construire la cible

la commande make

- s'assure que les cibles passées en argument (la première du makefile par défaut) sont *à jour*.
- *être à jour* c'est être plus récent que toutes ses dépendances
- *contruire* une cible, c'est executer les commandes qui permettent de la mettre à jour
- la construction s'arrête si
  - une dépendance ne peut être construite
  - une commande ne s'est pas executée correctement ($?)

syntaxe

```
# comments

targets: dependencies; modifiers commands

targets: dependencies
    modifiers commands

targets: dependencies \
other-dependencies
    a-very-long command with \
    a lot of arguments
```

remarques sur la syntaxe

- Attention: les commandes sont indentées avec une tabulation
- Tip: utilisez la coloration syntaxique de vim
- les \ protègent tous les caractères, fin de ligne y compris
- les $ du shell doivent être doublées dans les commandes

hello world, makefile

```
# $$USER instead of $USER
# $$HOME instead of $HOME
# shell command variables needs two
# ...

hello.txt:
        false
        echo greetings, $$USER
        touch hello.txt
```

hello world, make produit

```
false
makefile:2: recipe for target 'hello.txt' failed
make: *** [all] Error 1
```

modifiers de commande

- `@` annule l'echo
- `-` la commande est validée même en cas d'échec

```
hello.txt:
        @- false
        @ echo greetings, $$USER
        touch hello.txt
```

Exercice

- retranscrire la règle suivante ou une règle analogue
- ajoutez et supprimez des modifiers et assurez-vous d'en avoir compris le fonctionnement
- lancez make jusqu'à ce qu'il vous indique que tout est à jour.

plusieur règles

```
hello.txt:
        @- false
        @ echo greetings, $$USER
        touch hello.txt

hello2.txt:
        @- false
        @ echo greetings, $$USER
        touch hello.txt
```

QUESTION: que fait la commande `make` ?

réponse . . .

```
make              # construit hello.txt
make hello.txt    # construit hello.txt
make hello2.txt   # construit hello2.txt
```

factoriser

```
hello.txt hello2.txt:
	@- false
	@ echo greetings, $$USER
	touch $@
```

- target par default: hello.txt
- $@ est une variable automatique

variables automatiques

- y'en a plein!
- celles à connaitre sont
  - `$@` (la cible)
  - `$<` (la première dépendance). facile à retenir: pensez au redirecteur d'entrée (<)

usage des variables automatiques

```
behave.js:      behave.ls       ; lsc $<
components.js:  components.ls   ; lsc $<
bus.js:         bus.ls          ; lsc $<

index.html: index.md template.html
    pandoc \
    -t html+pipe_tables+grid_tables+multiline_tables+escape
    --toc -B menu --template -o $@ $<
```

pattern matching

```
behave.js:      behave.ls       ; lsc $<
components.js:  components.ls   ; lsc $<
bus.js:         bus.ls          ; lsc $<
```

peut s'écrire

```
%.js: %.ls ; lsc $<
```

stem

pattern match

- une règle peut trouver une cible avec
- un prefixe (fixe et optionnel)
- un “stem” (%)
- un suffixe (fixe et optionnel)
- le stem trouvé est disponible
- par % dans les dépendances
- par la variable $* dans la construction

Exercice

soit cible `funk-%-funky` dans votre `makefile`. vous écrirez le reste de la règle telle que

```
make funk-to-funky
```

soit une copie de `ashes-to-ashes` que vous aurez préalablement créé. au passage, vous affichez le stem sur sur la stdout.

variables

affectation	=
bind	:=
valeur par defaut	?=

variables (exemple)

```
template= template.beamer.latex
chapters = \
    index.md \
    touch.md
components= $(chapters) $(template) prelude.latex deps.pdf

mk-slides = pandoc \
    -f markdown -t beamer+escaped_line_breaks \
    --template $(template)

slides.pdf slides.latex: $(components)
    cat $(chapters) | $(mk-slides) -o $@
    @echo DONE
```

make dans vim

```
set make
set makeprg=make %:r.html
make
```

make dans vim + tmux

vim ~/.zshenv

```
promise/tmux/tailf/open () {
    tmux split-window -d  -p 20 \
        'tail -f ${PROMOUT:=/tmp/promise.mix}' }
promise/new () {
    : ${1:=zsh}
    "$@" &>> ${PROMOUT:=/tmp/promise.mix} & }
```

vim ~/.tmux.conf

```
bind-key P run-shell "zsh -c promise/tmux/tailf/open"
```

dans vim

```
:set makeprg=promise/new\ make
```

Questions?

merci . . .