Prise en main du projet GOCass

Projet 2ème partie - Séances 1 et 2

Amélie LEDEIN et Louis LEMONNIER

Et le sujet du projet est...

Programmer en OCaml un compilateur d'un sous-ensemble du langage Go [1], nommé ici Petit Go, vers de l'assembleur x86-64.

1. Go est un langage inspiré de C et Pascal `https://golang.org/doc/`.

Vous avez dit un compilateur ?

- Un **compilateur** est un programme qui lit un flux d'entrée structuré par une grammaire et qui produit une sortie.

2. ou **lexing** en anglais
3. ou **parsing** en anglais

Vous avez dit un compilateur ?

- Un **compilateur** est un programme qui lit un flux d'entrée structuré par une grammaire et qui produit une sortie.
- Il est capable de :
  - signaler des erreurs de syntaxe, de typage, de sémantique, etc ;
  - faire des optimisations (vitesse d'exécution, taille du code, utilisation de la mémoire, etc.).

2. ou **lexing** en anglais
3. ou **parsing** en anglais

Vous avez dit un compilateur ?

- Un **compilateur** est un programme qui lit un flux d'entrée structuré par une grammaire et qui produit une sortie.
- Il est capable de :
  - signaler des erreurs de syntaxe, de typage, de sémantique, etc ;
  - faire des optimisations (vitesse d'exécution, taille du code, utilisation de la mémoire, etc.).
- Les **étapes de compilation** sont :

Langage $\mathcal{L}_{input}$	$\Longrightarrow$	Analyse lexicale	$\Longrightarrow$	$\{(t_0, v_0), (t_1, v_1), ...\}$
$\{(t_0, v_0), (t_1, v_1), ...\}$	$\Longrightarrow$	Analyse grammaticale	$\Longrightarrow$	AST
AST	$\Longrightarrow$	Analyse sémantique	$\Longrightarrow$	Type abstrait
Type abstrait	$\Longrightarrow$	Génération du code	$\Longrightarrow$	Langage $\mathcal{L}_{output}$

- t_i = un **jeton** (ou **token** en anglais)
- v_i = une **valeur**
- AST : Abstract Syntax Tree, i.e. arbre de syntaxe abstraite
- Analyse syntaxique = Analyse lexicale [2] + Analyse grammaticale [3]
- ATTENTION : Certains ouvrages parlent d'analyse syntaxique au lieu d'analyse grammaticale.

2. ou **lexing** en anglais
3. ou **parsing** en anglais

Plus précisément, le sujet est...

Code Petit Go	$\Longrightarrow$	Analyse lexicale	$\Longrightarrow$	$\{(t_0, v_0), (t_1, v_1), ...\}$
$\{(t_0, v_0), (t_1, v_1), ...\}$	$\Longrightarrow$	Analyse grammaticale	$\Longrightarrow$	AST
AST	$\Longrightarrow$	Vérification du typage	$\Longrightarrow$	TAST
TAST	$\Longrightarrow$	Réécriture	$\Longrightarrow$	TAST transformé
TAST transformé	$\Longrightarrow$	Génération du code	$\Longrightarrow$	Code Assembleur

- AST : Abstract Syntax Tree
- TAST : Typed Abstract Syntax Tree

- Analyse syntaxique déjà codée pour vous.
- Analyse sémantique[4] = Typage + Réécriture
- Typage (Premier rendu)
- Réécriture + Génération de code (Deuxième rendu)

4. L'analyse sémantique étudie l'arbre de syntaxe abstraite produit par l'analyse syntaxique pour éliminer au maximum les programmes qui ne sont pas corrects du point de vue de la sémantique.

Planning

Séance 1 - mardi 26 octobre - Amélie LEDEIN

Séance 2 - mercredi 27 ou jeudi 28 octobre - Louis LEMONNIER

Séance 3 - vendredi 29 octobre - Amélie LEDEIN

Vacances de la Toussaint (1 semaine)

Séance 4 - mercredi 10 novembre (**Classe entière**)

Séance 5 - 17 ou 18 novembre

Séance 6 - 24 ou 25 novembre

Rendu **Lundi 29 novembre 2021 à 23h59 : Typage**

Séance 7 - mercredi 1er décembre (**Classe entière**)

Séance 8 - 8 ou 9 décembre

Séance 9 - 15 ou 16 décembre

Vacances de Noël (2 semaines)

Rendu **Lundi 3 janvier 2022 à 23h59 : Réécriture + Génération de code**

Oral Semaine du 3 janvier - Soutenance de 15 minutes

Sommaire

Exercice 1 - Comprendre la grammaire du Petit Go

- Lisez l'exemple dans la partie "Syntaxe" du sujet.
- Lisez la grammaire disponible à l'annexe A.2 - Figure 1 du sujet.
- Ecrivez une fonction écrite dans le langage de Petit Go qui calcule la somme de tous les entiers de 1 à n.

3 volontaires pour le faire au tableau !

Sommaire

Rappels de l'objectif

- **Objectif : S'assurer de la conformité du typage.**
- La grammaire du Petit Go est disponible dans le sujet.
- Elle est représentée en OCaml par le type `pfile`.
- Le système de types de Petit Go est un système très simple, sans relation de sous-typage entre les types.

Typage statique

- **Type** : $\tau ::= \texttt{int} \mid \texttt{bool} \mid \texttt{string} \mid S \mid *\tau$
  où S désigne un nom de structure (introduit avec la déclaration `type`).
- **Déclaration de variable** : $\mathit{var\_decl} ::= x : \tau$
- **Déclaration de fonction** :
  $\mathit{fct\_decl} ::= f(\tau_1, \ldots, \tau_n) \Rightarrow \tau'_1, \ldots, \tau'_m$ avec $n \geq 0$ et $m \geq 0$
- **Déclaration de structure** :
  $\mathit{struct\_decl} ::= S\{\mathit{var\_decl}, ..., \mathit{var\_decl}\}$
- **Contexte de typage** :

$$\Gamma \quad ::= \quad \Gamma, \mathit{struct\_decl} \mid \Gamma, \mathit{fct\_decl} \mid \Gamma, \mathit{var\_decl} \mid \emptyset$$

Le fichier `src/tast.mli` code pour le moment que le contexte pour les variables.

Règles d'inférence

- Une **règle d'inférence** est de la forme :

$$(nom)\ \frac{premisse(s)}{conclusion}$$

où la conclusion est un **jugement**.

Règles d'inférence

- Une **règle d'inférence** est de la forme :

$$(nom)\ \frac{premisse(s)}{conclusion}$$

où la conclusion est un **jugement**.

- Jugements :
  - $\Gamma \vdash \tau\ bf$ signifie

    $\ll$ le type τ est bien formé dans le contexte Γ $\gg$
  - $\Gamma \vdash e : \tau$ signifie

    $\ll$ dans le contexte Γ, l'expression e est bien typée de type τ $\gg$
  - $\Gamma \vdash_l e : \tau$ signifie

    $\ll$ e est une valeur gauche (*l-value*) bien typée et de type τ $\gg$
  - $\Gamma \vdash f(e_1, \ldots, e_n) \Rightarrow \tau_1, \ldots, \tau_m$ signifie

    $\ll$ dans le contexte Γ, l'appel de fonction $f(e_1, \ldots, e_n)$ est bien typé et renvoie m valeurs de types $\tau_1, \ldots, \tau_m$ $\gg$
  - $\Gamma \vdash s$ signifie

    $\ll$ dans le contexte Γ, l'instruction s est bien typée $\gg$

Bonne formation d'un type

Rappel : $\tau ::= \texttt{int} \mid \texttt{bool} \mid \texttt{string} \mid S \mid *\tau$

Bonne formation d'un type

Rappel : $\tau ::= \texttt{int} \mid \texttt{bool} \mid \texttt{string} \mid S \mid *\tau$

$$(BF_{int})\ \frac{}{\Gamma \vdash \texttt{int}\ bf} \qquad (BF_{bool})\ \frac{}{\Gamma \vdash \texttt{bool}\ bf}$$

$$(BF_{string})\ \frac{}{\Gamma \vdash \texttt{string}\ bf}$$

$$(BF_{struct})\ \frac{S \in \Gamma}{\Gamma \vdash S\ bf} \qquad (BF_{pointeur})\ \frac{\Gamma \vdash \tau\ bf}{\Gamma \vdash *\tau\ bf}$$

où le jugement $\Gamma \vdash \tau\ bf$ signifie $\ll$ le type τ est bien formé dans le contexte $\Gamma \gg$.

Typage d'une expression (1)

$$(TE_{const})\frac{\text{Constante } c \ \in \Sigma \text{ de type } \tau}{\Gamma \vdash c : \tau}$$

$$(TE_{moins})\ \frac{\Gamma \vdash e : \texttt{int}}{\Gamma \vdash \texttt{-}\ e : \texttt{int}} \qquad (TE_{neg})\ \frac{\Gamma \vdash e : \texttt{bool}}{\Gamma \vdash \texttt{!}e : \texttt{bool}}$$

$$(TE_{eq})\ \frac{\Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \tau \quad op \in \{\texttt{==},\ \texttt{!=}\} \quad e_1 \neq \texttt{nil} \vee e_2 \neq \texttt{nil}}{\Gamma \vdash e_1\ op\ e_2 : \texttt{bool}}$$

$$(TE_{lin})\ \frac{\Gamma \vdash e_1 : \texttt{int} \quad \Gamma \vdash e_2 : \texttt{int} \quad op \in \{\texttt{<},\ \texttt{<=},\ \texttt{>},\ \texttt{>=}\}}{\Gamma \vdash e_1\ op\ e_2 : \texttt{bool}}$$

$$(TE_{a})\ \frac{\Gamma \vdash e_1 : \texttt{int} \quad \Gamma \vdash e_2 : \texttt{int} \quad op \in \{\texttt{+},\ \texttt{-},\ \texttt{*},\ \texttt{/},\ \texttt{\%}\}}{\Gamma \vdash e_1\ op\ e_2 : \texttt{int}}$$

$$(TE_{b})\ \frac{\Gamma \vdash e_1 : \texttt{bool} \quad \Gamma \vdash e_2 : \texttt{bool} \quad op \in \{\texttt{\&\&},\ \texttt{||}\}}{\Gamma \vdash e_1\ op\ e_2 : \texttt{bool}}$$

où le jugement $\Gamma \vdash e : \tau$ signifie « dans le contexte Γ, l'expression e est bien typée de type τ ».

Typage d'une expression (1)

$$(TE_{const})\frac{\text{Constante } c \ \in \Sigma \text{ de type } \tau}{\Gamma \vdash c : \tau}$$

$$(TE_{moins})\ \frac{\Gamma \vdash e : \texttt{int}}{\Gamma \vdash \texttt{-}\ e : \texttt{int}} \qquad (TE_{neg})\ \frac{\Gamma \vdash e : \texttt{bool}}{\Gamma \vdash \texttt{!}e : \texttt{bool}}$$

$$(TE_{eq})\ \frac{\Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \tau \quad op \in \{\texttt{==},\ \texttt{!=}\} \quad e_1 \neq \texttt{nil} \vee e_2 \neq \texttt{nil}}{\Gamma \vdash e_1\ op\ e_2 : \texttt{bool}}$$

$$(TE_{lin})\ \frac{\Gamma \vdash e_1 : \texttt{int} \quad \Gamma \vdash e_2 : \texttt{int} \quad op \in \{\texttt{<},\ \texttt{<=},\ \texttt{>},\ \texttt{>=}\}}{\Gamma \vdash e_1\ op\ e_2 : \texttt{bool}}$$

$$(TE_{a})\ \frac{\Gamma \vdash e_1 : \texttt{int} \quad \Gamma \vdash e_2 : \texttt{int} \quad op \in \{\texttt{+},\ \texttt{-},\ \texttt{*},\ \texttt{/},\ \texttt{\%}\}}{\Gamma \vdash e_1\ op\ e_2 : \texttt{int}}$$

$$(TE_{b})\ \frac{\Gamma \vdash e_1 : \texttt{bool} \quad \Gamma \vdash e_2 : \texttt{bool} \quad op \in \{\texttt{\&\&},\ \texttt{||}\}}{\Gamma \vdash e_1\ op\ e_2 : \texttt{bool}}$$

Exercice 2 - Démontrer que l'expression suivante a bien le type attendu : `4 + 75 >= 100 || 3 == 42 && true`.

Typage d'une expression (2)

$$\frac{\Gamma \vdash_l e : \tau' \quad \Gamma \vdash e.x : \tau}{\Gamma \vdash_l e.x : \tau} \qquad \frac{x : \tau \in \Gamma}{\Gamma \vdash_l x : \tau} \qquad \frac{\Gamma \vdash_l e : \tau \quad e \neq \_}{\Gamma \vdash e : \tau}$$

$$\frac{\Gamma \vdash e : S \quad \Gamma \ni S\{\, x : \tau \,\}}{\Gamma \vdash e.x : \tau} \qquad \frac{\Gamma \vdash e : *S \quad e \neq \texttt{nil} \quad \Gamma \ni S\{\, x : \tau \,\}}{\Gamma \vdash e.x : \tau}$$

$$\frac{\Gamma \vdash \tau \ bf}{\Gamma \vdash \texttt{nil} : *\tau} \qquad \frac{\Gamma \vdash_l e : \tau}{\Gamma \vdash \&e : *\tau}$$

$$\frac{\Gamma \vdash e : *\tau \quad e \neq \texttt{nil}}{\Gamma \vdash_l *e : \tau} \qquad \frac{S \in \Gamma}{\Gamma \vdash \texttt{new}(S) : *S}$$

où le jugement $\Gamma \vdash_l e : \tau$ signifie « e est une valeur gauche (*l-value*) bien typée et de type τ ».

Typage d'une expression (2)

$$\frac{\Gamma \vdash_l e : \tau' \quad \Gamma \vdash e.x : \tau}{\Gamma \vdash_l e.x : \tau} \qquad \frac{x : \tau \in \Gamma}{\Gamma \vdash_l x : \tau} \qquad \frac{\Gamma \vdash_l e : \tau \quad e \neq \_}{\Gamma \vdash e : \tau}$$

$$\frac{\Gamma \vdash e : S \quad \Gamma \ni S\{\, x : \tau \,\}}{\Gamma \vdash e.x : \tau} \qquad \frac{\Gamma \vdash e : *S \quad e \neq \texttt{nil} \quad \Gamma \ni S\{\, x : \tau \,\}}{\Gamma \vdash e.x : \tau}$$

$$\frac{\Gamma \vdash \tau \ bf}{\Gamma \vdash \texttt{nil} : *\tau} \qquad \frac{\Gamma \vdash_l e : \tau}{\Gamma \vdash \&e : *\tau}$$

$$\frac{\Gamma \vdash e : *\tau \quad e \neq \texttt{nil}}{\Gamma \vdash_l *e : \tau} \qquad \frac{S \in \Gamma}{\Gamma \vdash \texttt{new}(S) : *S}$$

Exercice 3 - Démontrer que l'expression suivante a bien le type attendu, avec $\Gamma = S_{coord}\{x : int, y : int\} : e.x$

Typage d'un appel

$$\frac{\Gamma \vdash f(e_1, \ldots, e_n) \Rightarrow \tau_1}{\Gamma \vdash f(e_1, \ldots, e_n) : \tau_1}$$

$$\frac{\Gamma \ni f(\tau_1, \ldots, \tau_n) \Rightarrow \tau'_1, \ldots, \tau'_m \quad \forall i,\ \Gamma \vdash e_i : \tau_i}{\Gamma \vdash f(e_1, \ldots, e_n) \Rightarrow \tau'_1, \ldots, \tau'_m}$$

$$\frac{n \geq 2 \quad \Gamma \ni f(\tau_1, \ldots, \tau_n) \Rightarrow \tau'_1, \ldots, \tau'_m \quad \Gamma \vdash g(e_1, \ldots, e_k) \Rightarrow \tau_1, \ldots, \tau_n}{\Gamma \vdash f(g(e_1, \ldots, e_k)) \Rightarrow \tau'_1, \ldots, \tau'_m}$$

Cette dernière règle permet de passer directement les n résultats d'une fonction g en arguments d'une fonction f.

où le jugement $\Gamma \vdash f(e_1, \ldots, e_n) \Rightarrow \tau_1, \ldots, \tau_m$ signifie « dans le contexte Γ, l'appel de fonction $f(e_1, \ldots, e_n)$ est bien typé et renvoie m valeurs de types $\tau_1, \ldots, \tau_m$ ».

Typage d'une instruction (1)

$$\frac{\Gamma \vdash_l e : \texttt{int}}{\Gamma \vdash e\texttt{++}} \qquad \frac{\Gamma \vdash_l e : \texttt{int}}{\Gamma \vdash e\texttt{--}}$$

$$\frac{\forall i,\ \Gamma \vdash e_i : \tau_i}{\Gamma \vdash \texttt{fmt.Print}(e_1, \ldots, e_n)} \qquad \frac{n \geq 2 \quad \Gamma \vdash f(e_1, \ldots, e_k) \Rightarrow \tau_1, \ldots, \tau_n}{\Gamma \vdash \texttt{fmt.Print}(f(e_1, \ldots, e_k))}$$

$$\frac{\forall i,\ \Gamma \vdash_l e_i : \tau_i \quad \forall i,\ \Gamma \vdash e'_i : \tau_i}{\Gamma \vdash e_1, \ldots, e_n = e'_1, \ldots, e'_n} \qquad \frac{\forall i,\ \Gamma \vdash_l e_i : \tau_i \quad \Gamma \vdash f(e'_1, \ldots, e'_m) \Rightarrow}{\Gamma \vdash e_1, \ldots, e_n = f(e'_1, \ldots, e'_m)}$$

$$\frac{\Gamma \vdash e : \texttt{bool} \quad \Gamma \vdash b_1 \quad \Gamma \vdash b_2}{\Gamma \vdash \texttt{if}\ e\ b_1\ \texttt{else}\ b_2} \qquad \frac{\Gamma \vdash e : \texttt{bool} \quad \Gamma \vdash b}{\Gamma \vdash \texttt{for}\ e\ b}$$

$$\frac{\forall i,\ \Gamma \vdash e_i : \tau_i}{\Gamma \vdash \texttt{return}\ e_1, \ldots, e_n} \qquad \frac{\Gamma \vdash f(e_1, \ldots, e_k) \Rightarrow \tau_1, \ldots, \tau_n}{\Gamma \vdash \texttt{return}\ f(e_1, \ldots, e_k)}$$

Typage d'une instruction (2)

$$\frac{\Gamma \vdash \tau \ bf \quad \Gamma + x_1 : \tau, \ldots, x_n : \tau \vdash \{s_2; \ldots; s_m\}}{\Gamma \vdash \{\texttt{var } x_1, \ldots, x_n \ \tau; s_2; \ldots; s_m\}}$$

$$\frac{\Gamma \vdash \tau \ bf \quad \forall i,\ \Gamma \vdash e_i : \tau \quad \Gamma + x_1 : \tau, \ldots, x_n : \tau \vdash \{s_2; \ldots; s_n\}}{\Gamma \vdash \{\texttt{var } x_1, \ldots, x_n \ \tau = e_1, \ldots, e_n; s_2; \ldots; s_n\}}$$

$$\frac{\forall i,\ e_i \neq \texttt{nil} \quad \forall i,\ \Gamma \vdash e_i : \tau_i \quad \Gamma + x_1 : \tau_1, \ldots, x_n : \tau_n \vdash \{s_2; \ldots; s_m\}}{\Gamma \vdash \{\texttt{var } x_1, \ldots, x_n = e_1, \ldots, e_n; s_2; \ldots; s_m\}}$$

$$\frac{\Gamma \vdash \tau \ bf \quad \Gamma \vdash f(e_1, \ldots, e_k) \Rightarrow \tau^n \quad \Gamma + x_1 : \tau, \ldots, x_n : \tau \vdash \{s_2; \ldots; s_m\}}{\Gamma \vdash \{\texttt{var } x_1, \ldots, x_n \ \tau = f(e_1, \ldots, e_k); s_2; \ldots; s_m\}}$$

$$\frac{\Gamma \vdash f(e_1, \ldots, e_k) \Rightarrow \tau_1, \ldots, \tau_n \quad \Gamma + x_1 : \tau_1, \ldots, x_n : \tau_n \vdash \{s_2; \ldots; s_m\}}{\Gamma \vdash \{\texttt{var } x_1, \ldots, x_n = f(e_1, \ldots, e_k); s_2; \ldots; s_m\}}$$

$$\frac{}{\Gamma \vdash \{\}} \qquad \frac{\Gamma \vdash s_1 \quad \Gamma \vdash \{s_2; \ldots; s_n\}}{\Gamma \vdash \{s_1; \ldots; s_n\}}$$

Par ailleurs, toutes les variables introduites dans un *même* bloc doivent porter des noms différents. Une exception est faite pour la variable « _ ». où le jugement $\Gamma \vdash s$ signifie « dans le contexte Γ, instruction s est bien typée ».

Typage d'une instruction (2)

$$\frac{\Gamma \vdash \tau \ bf \quad \Gamma + x_1 : \tau, \ldots, x_n : \tau \vdash \{s_2; \ldots; s_m\}}{\Gamma \vdash \{\texttt{var } x_1, \ldots, x_n \ \tau; s_2; \ldots; s_m\}}$$

$$\frac{\Gamma \vdash \tau \ bf \quad \forall i,\ \Gamma \vdash e_i : \tau \quad \Gamma + x_1 : \tau, \ldots, x_n : \tau \vdash \{s_2; \ldots; s_n\}}{\Gamma \vdash \{\texttt{var } x_1, \ldots, x_n \ \tau = e_1, \ldots, e_n; s_2; \ldots; s_n\}}$$

$$\frac{\forall i,\ e_i \neq \texttt{nil} \quad \forall i,\ \Gamma \vdash e_i : \tau_i \quad \Gamma + x_1 : \tau_1, \ldots, x_n : \tau_n \vdash \{s_2; \ldots; s_m\}}{\Gamma \vdash \{\texttt{var } x_1, \ldots, x_n = e_1, \ldots, e_n; s_2; \ldots; s_m\}}$$

$$\frac{\Gamma \vdash \tau \ bf \quad \Gamma \vdash f(e_1, \ldots, e_k) \Rightarrow \tau^n \quad \Gamma + x_1 : \tau, \ldots, x_n : \tau \vdash \{s_2; \ldots; s_m\}}{\Gamma \vdash \{\texttt{var } x_1, \ldots, x_n \ \tau = f(e_1, \ldots, e_k); s_2; \ldots; s_m\}}$$

$$\frac{\Gamma \vdash f(e_1, \ldots, e_k) \Rightarrow \tau_1, \ldots, \tau_n \quad \Gamma + x_1 : \tau_1, \ldots, x_n : \tau_n \vdash \{s_2; \ldots; s_m\}}{\Gamma \vdash \{\texttt{var } x_1, \ldots, x_n = f(e_1, \ldots, e_k); s_2; \ldots; s_m\}}$$

$$\frac{}{\Gamma \vdash \{\}} \qquad \frac{\Gamma \vdash s_1 \quad \Gamma \vdash \{s_2; \ldots; s_n\}}{\Gamma \vdash \{s_1; \ldots; s_n\}}$$

où le jugement $\Gamma \vdash s$ signifie « dans le contexte Γ, instruction s est bien typée ».

Exercice 4 : Démontrer que les 2 programmes écrits à l'exercice 1 ont le type attendu.

Sommaire

Hiérarchie générale des fichiers

Code Petit Go	$\Longrightarrow$	Analyse lexicale	$\Longrightarrow$	$\{(t_0, v_0), (t_1, v_1), ...\}$
$\{(t_0, v_0), (t_1, v_1), ...\}$	$\Longrightarrow$	Analyse grammaticale	$\Longrightarrow$	AST
AST	$\Longrightarrow$	Vérification du typage	$\Longrightarrow$	TAST
TAST	$\Longrightarrow$	Réécriture	$\Longrightarrow$	TAST transformé
TAST transformé	$\Longrightarrow$	Génération du code	$\Longrightarrow$	AA

- AST : Abstract Syntax Tree
- TAST : Typed Abstract Syntax Tree
- AA : Abstract Assembly

Hiérarchie générale des fichiers

Code Petit Go	$\Longrightarrow$	`lexer.mll`	$\Longrightarrow$	$\{(t_0, v_0), (t_1, v_1), ...\}$
$\{(t_0, v_0), (t_1, v_1), ...\}$	$\Longrightarrow$	`parser.mly`	$\Longrightarrow$	AST
AST	$\Longrightarrow$	`typing.ml`	$\Longrightarrow$	TAST
TAST	$\Longrightarrow$	`rewrite.ml`	$\Longrightarrow$	TAST transformé
TAST transformé	$\Longrightarrow$	`compile.ml`	$\Longrightarrow$	AA

- AST : Abstract Syntax Tree
- TAST : Typed Abstract Syntax Tree
- AA : Abstract Assembly

Hiérarchie générale des fichiers

Code Petit Go	$\Longrightarrow$	`lexer.mll`	$\Longrightarrow$	$\{(t_0, v_0), (t_1, v_1), ...\}$
$\{(t_0, v_0), (t_1, v_1), ...\}$	$\Longrightarrow$	`parser.mly`	$\Longrightarrow$	AST
AST	$\Longrightarrow$	`typing.ml`	$\Longrightarrow$	TAST
TAST	$\Longrightarrow$	`rewrite.ml`	$\Longrightarrow$	TAST transformé
TAST transformé	$\Longrightarrow$	`compile.ml`	$\Longrightarrow$	AA

- AST : Abstract Syntax Tree $\rightarrow$ `ast.mli`
- TAST : Typed Abstract Syntax Tree $\rightarrow$ `tast.mli`
- AA : Abstract Assembly $\rightarrow$ `x86_64.ml` et `x86_64.mli`

Hiérarchie générale des fichiers

Code Petit Go	$\Longrightarrow$	`lexer.mll`	$\Longrightarrow$	$\{(t_0, v_0), (t_1, v_1), ...\}$
$\{(t_0, v_0), (t_1, v_1), ...\}$	$\Longrightarrow$	`parser.mly`	$\Longrightarrow$	AST
AST	$\Longrightarrow$	`typing.ml`	$\Longrightarrow$	TAST
TAST	$\Longrightarrow$	`rewrite.ml`	$\Longrightarrow$	TAST transformé
TAST transformé	$\Longrightarrow$	`compile.ml`	$\Longrightarrow$	AA

- AST : Abstract Syntax Tree → `ast.mli`
- TAST : Typed Abstract Syntax Tree → `tast.mli`
- AA : Abstract Assembly → `x86_64.ml` et `x86_64.mli`

- `lib.ml` : Extension de la bibliothèque standard de OCaml
- `pretty.ml` : Affichage des AST et des TAST
- `main.ml` : Programme général et parsing de la ligne de commande (option `--debug`, `--parse-only` et `--type-only`)

Hiérarchie générale des fichiers

Code Petit Go	$\Longrightarrow$	`lexer.mll`	$\Longrightarrow$	$\{(t_0, v_0), (t_1, v_1), ...\}$
$\{(t_0, v_0), (t_1, v_1), ...\}$	$\Longrightarrow$	`parser.mly`	$\Longrightarrow$	AST
AST	$\Longrightarrow$	`typing.ml`	$\Longrightarrow$	TAST
TAST	$\Longrightarrow$	`rewrite.ml`	$\Longrightarrow$	TAST transformé
TAST transformé	$\Longrightarrow$	`compile.ml`	$\Longrightarrow$	AA

- AST : Abstract Syntax Tree → `ast.mli`
- TAST : Typed Abstract Syntax Tree → `tast.mli`
- AA : Abstract Assembly → `x86_64.ml` et `x86_64.mli`

- `lib.ml` : Extension de la bibliothèque standard de OCaml
- `pretty.ml` : Affichage des AST et des TAST
- `main.ml` : Programme général et parsing de la ligne de commande (option `--debug`, `--parse-only` et `--type-only`)

- Pour lancer la compilation : `Makefile`
- Pour gérer la compilation : `dune` et `dune-project`

Quelques outils pour vous aider

Quelques outils pour vous aider

- Dune : `https://dune.readthedocs.io/`

Quelques outils pour vous aider

- Dune : `https://dune.readthedocs.io/`
- Makefile :
  - La commande `make` crée le compilateur nommé pgoc.
  - La commande `make clean` efface tous les fichiers que `make` a engendrés et ne laisse dans le répertoire que les fichiers sources.
  - Le fichier `Makefile` distribué utilise `dune`, mais vous pouvez le réécrire pour éviter l'installation de ce paquet.

Quelques outils pour vous aider

- Dune : `https://dune.readthedocs.io/`
- Makefile :
  - La commande `make` crée le compilateur nommé pgoc.
  - La commande `make clean` efface tous les fichiers que `make` a engendrés et ne laisse dans le répertoire que les fichiers sources.
  - Le fichier `Makefile` distribué utilise `dune`, mais vous pouvez le réécrire pour éviter l'installation de ce paquet.
- Git : Ce que nous allons voir dès maintenant !

Sommaire

Qu'est-ce qu'un système de gestion de version ?

Qu'est-ce qu'un système de gestion de version ?

- Système logiciel qui permet de maintenir et de gérer toutes les versions d'un ensemble de fichiers.

Qu'est-ce qu'un système de gestion de version ?

- Système logiciel qui permet de maintenir et de gérer toutes les versions d'un ensemble de fichiers.
- Pourquoi utiliser un système de gestion de version ?
  - Revenir facilement à une version précédente.
  - Suivre l'évolution du projet dans le temps.
  - Permettre le travail en parallèle sur des parties disjointes du projet et gérer les modifications concurrentes.
  - Faciliter la détection et la correction des erreurs.
  - etc.

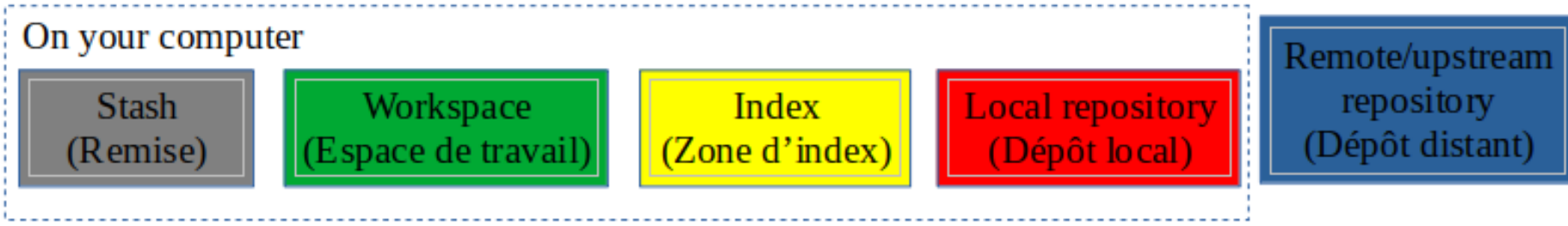
Qu'est-ce qu'un système de gestion de version ?

- Système logiciel qui permet de maintenir et de gérer toutes les versions d'un ensemble de fichiers.
- Pourquoi utiliser un système de gestion de version ?
  - Revenir facilement à une version précédente.
  - Suivre l'évolution du projet dans le temps.
  - Permettre le travail en parallèle sur des parties disjointes du projet et gérer les modifications concurrentes.
  - Faciliter la détection et la correction des erreurs.
  - etc.
- Un exemple :

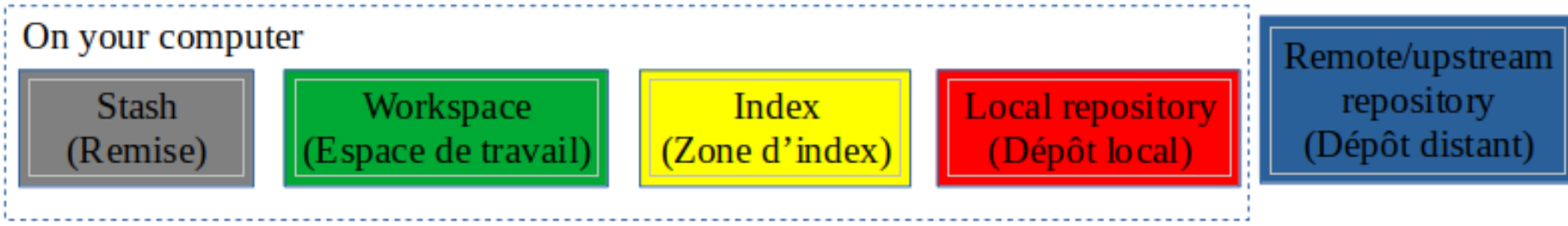
$\rightarrow$ Un système de contrôle de version distribué (DVCS) gratuit et open source

$\rightarrow$ Conçu pour gérer tous les projets, des plus petits aux plus grands, avec rapidité et efficacité.

Git plus en détail

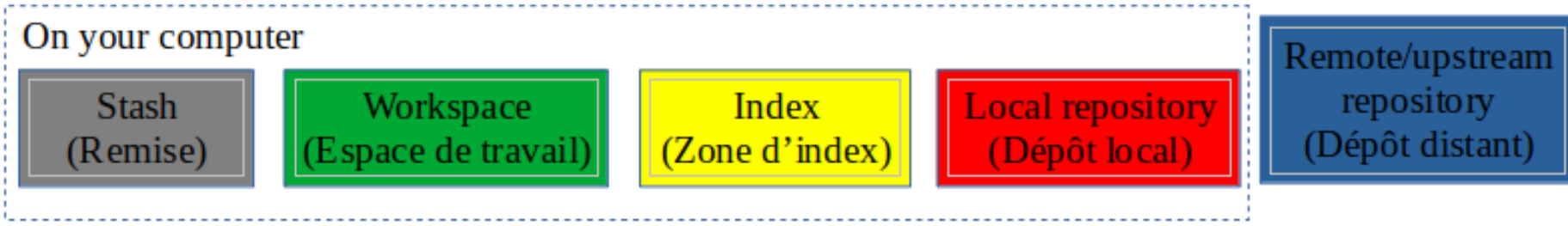


Git plus en détail



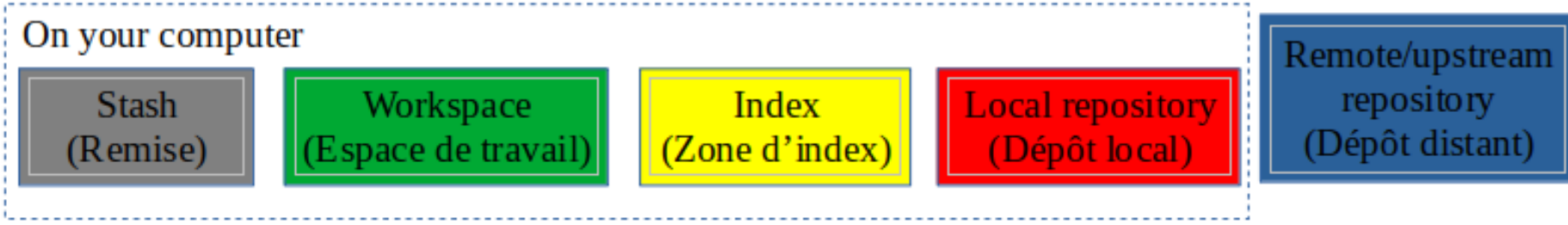
- **Stash** (**remise**) : Un endroit pour cacher les modifications pendant que vous travaillez sur autre chose.

Git plus en détail



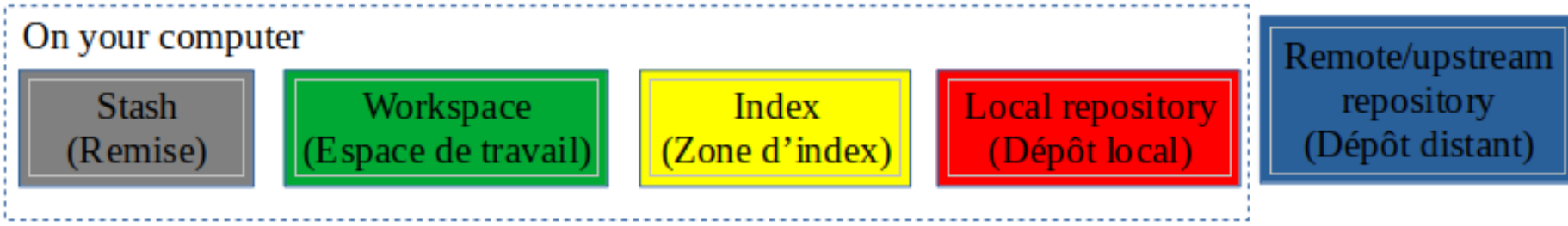
- **Stash** (**remise**) : Un endroit pour cacher les modifications pendant que vous travaillez sur autre chose.
- **Workspace** (**espace de travail**) : Version locale des fichiers.

Git plus en détail



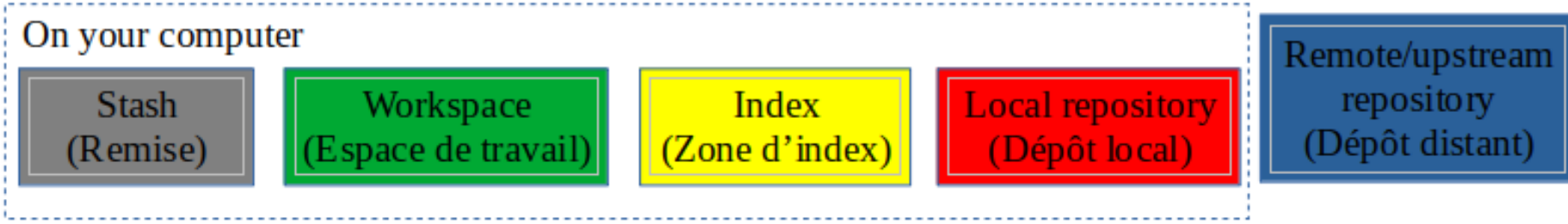
- **Stash** (**remise**) : Un endroit pour cacher les modifications pendant que vous travaillez sur autre chose.
- **Workspace** (**espace de travail**) : Version locale des fichiers.
- **Index** (**zone d'index**), **staging area**, **staged files** or (current directory) cache : Les fichiers que vous voulez livrer. Avant de "livrer" (archiver) des fichiers, vous devez d'abord les ajouter à l'index.

Git plus en détail

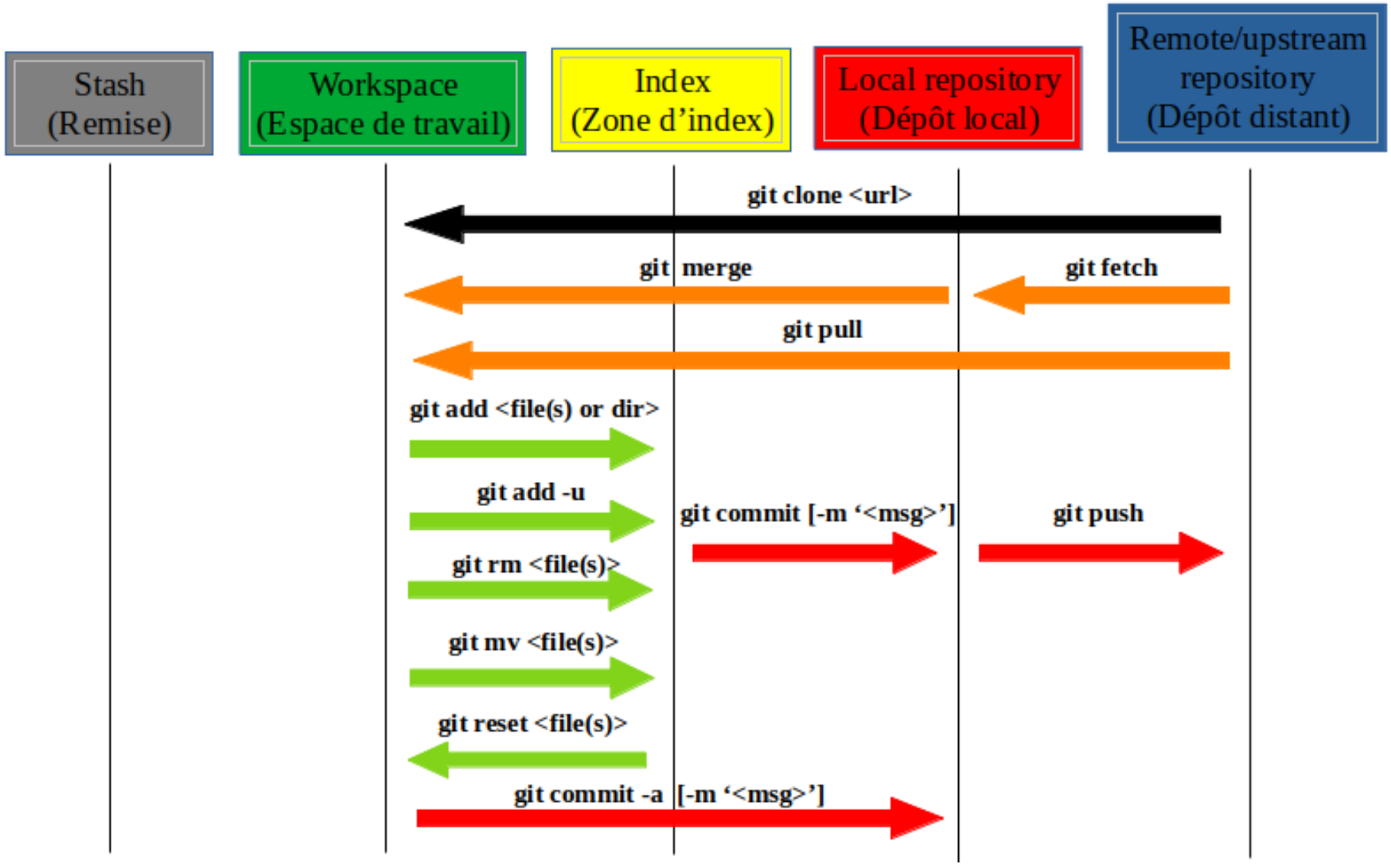


- **Stash** (**remise**) : Un endroit pour cacher les modifications pendant que vous travaillez sur autre chose.
- **Workspace** (**espace de travail**) : Version locale des fichiers.
- **Index** (**zone d'index**), **staging area**, **staged files** or (current directory) cache : Les fichiers que vous voulez livrer. Avant de "livrer" (archiver) des fichiers, vous devez d'abord les ajouter à l'index.
- **Local repository** (**dépôt local**) : Un sous-répertoire nommé `.git` qui contient tous les fichiers déposés nécessaires - un squelette de dépôt Git.

Git plus en détail



- **Stash** (**remise**) : Un endroit pour cacher les modifications pendant que vous travaillez sur autre chose.
- **Workspace** (**espace de travail**) : Version locale des fichiers.
- **Index** (**zone d'index**), **staging area**, **staged files** or (current directory) cache : Les fichiers que vous voulez livrer. Avant de "livrer" (archiver) des fichiers, vous devez d'abord les ajouter à l'index.
- **Local repository** (**dépôt local**) : Un sous-répertoire nommé `.git` qui contient tous les fichiers déposés nécessaires - un squelette de dépôt Git.
- **Remote/upstream repository** (**dépôt distant**) : Versions de votre projet qui sont hébergées sur Internet ou sur un réseau, garantissant que toutes vos modifications sont disponibles pour les autres développeurs.


Stash
(Remise)
Workspace
(Espace de travail)
Index
(Zone d'index)
Local repository
(Dépôt local)
Remote/upstream
repository
(Dépôt distant)
git clone <url>
git merge
git fetch
git pull
git add <file(s) or dir>
git add -u
git commit [-m '<msg>']
git push
git rm <file(s)>
git mv <file(s)>
git reset <file(s)>
git commit -a [-m '<msg>']

Synchronisation avec le répertoire distant

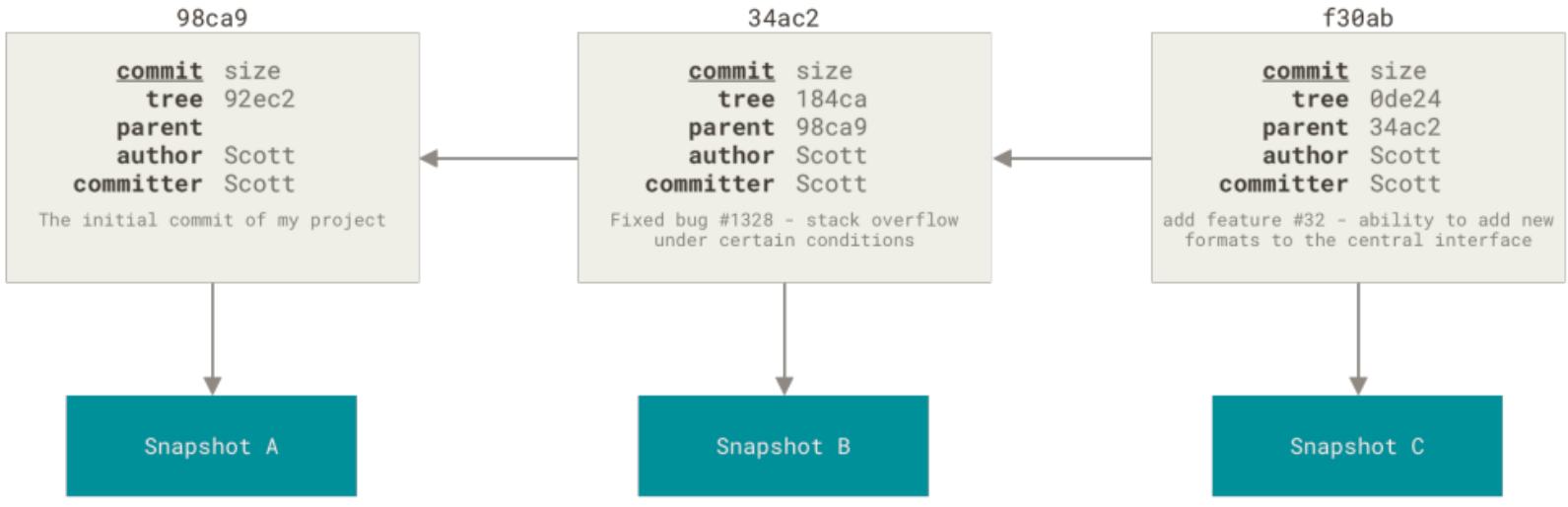
`$ git clone <url>`
Retrieve an entire repository from hosted location via URL.

`$ git fetch <alias>`
Fetch down all the branches from that Git remote.

`$ git merge <alias/<branch>`
Merge a remote branch into your current branch to bring it up to date.

`$ git pull`
Fetch and merge any commits from the tracking remote branch.

Commit


98ca9
commit size
tree 92ec2
parent
author Scott
committer Scott
The initial commit of my project
34ac2
commit size
tree 184ca
parent 98ca9
author Scott
committer Scott
Fixed bug #1328 - stack overflow under certain conditions
f30ab
commit size
tree 0de24
parent 34ac2
author Scott
committer Scott
add feature #32 - ability to add new formats to the central interface
Snapshot A
Snapshot B
Snapshot C

Faire des modifications

`$ git add <file(s)>`
Add a file (or several) as it looks now to your next commit (stage).

`$ git rm <file(s)>`
Delete the file (or several) from the project and stage the removal for commit.

`$ git mv <old-name-file> <new-name-file>`
Rename the file and stage the renaming.

`$ git mv <existing-path> <new-path>`
Change an existing file path and stage the move.

`$ git reset <file(s)>`
Unstage a file (or several) while retaining the changes in working directory.

`$ git commit [-m "<descriptive message>"]`
Commit your staged content as a new commit snapshot.

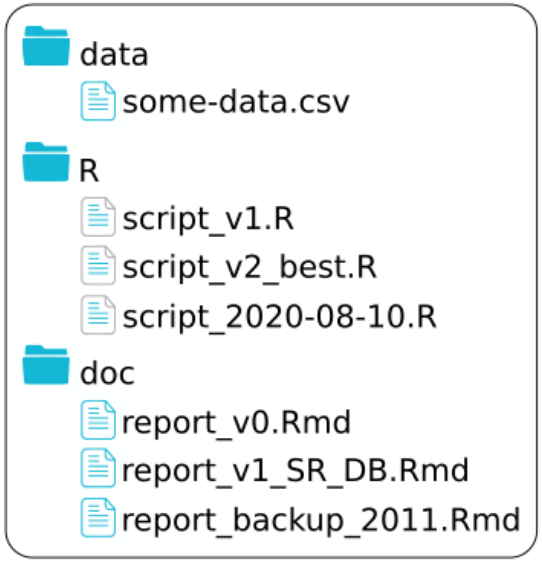
`$ git push <alias> <branch>`
Transmit local branch commits to the remote repository branch.

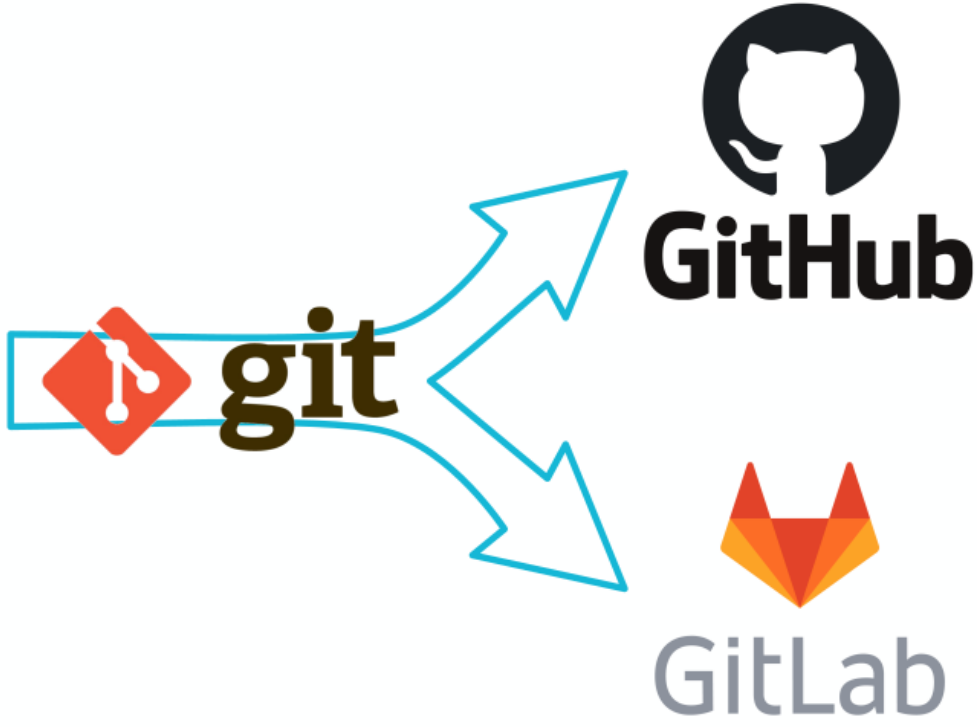
D'autres commandes utiles

- Setup :
  Configuring user information used across all local repositories.
  - `$ git config --global user.name "[firstname lastname]"`
    Set a name that is identifiable for credit when review version history.
  - `$ git config --global user.email "[valid-email]"`
    Set a email address that will be associated with each history marker.

  Note : `export EDITOR=emacs` (or `vim`, etc.)
  To configure correctly your editor with Git.
- To collect information :
  - `$ git status`
    Show modified files in working directory, staged for your next commit.
  - `$ git diff`
    Diff of what is changed but not staged.
  - `$ git diff --staged`
    Diff of what is staged but not yet committed.

Git et GitHub/GitLab





2 interfaces graphiques pour vous aider !

Exercice 5 - Prise en main de GitLab

1. Installez git.
2. Créez vous un compte sur GitLab.
3. Créez un nouveau répertoire.
4. Aller chercher les fichiers sources sur eCampus.
5. Vérifier que les sources compilent de votre côté avec `make`.
6. Ajoutez les fichiers sources du projet dans votre répertoire.

Sommaire

Consignes générales

Travail à faire :

- **seul.e.**
- à partir des sources initiales disponibles sur eCampus
- **en modifiant uniquement les** `TODO`.
  `TODO uncomment` signifie qu'il faut simplement dé-commenter le code déjà écrit.

Travail à rendre :

- **sur eCampus**
- sous forme d'une **archive** `tar` compressée (option "z" de `tar`), nommée *votre_nom*`.tgz`, contenant un répertoire, appelé *votre_nom*, où doivent se trouver les *sources* du compilateur (inutile d'inclure les fichiers compilés) incluant les fichiers sources distribués.
- contenant :
  - Les **sources**
  - Un **court rapport** (format ASCII, Markdown ou PDF) expliquant les différents choix techniques qui ont été faits, les difficultés rencontrées, les éléments réalisés et leur test, les éléments non réalisés et plus généralement toute différence par rapport à ce qui a été demandé.
  - Des **tests**.

Quelques conseils

- **Procéder construction par construction** et s'assurer à chaque étape que votre code compile et passe les tests donnés dans le répertoire `tests`.
- **Sauvegarder régulièrement** (Git ou support externe) afin de ne pas perdre des versions qui fonctionnent.
- Vous devez documenter votre code et écrire un journal de votre développement (le format Markdown est très facile pour cette tâche). Ces notes et commentaires vous permettrons de générer rapidement le rapport du projet.
- **Respectez le calendrier** et les étapes proposées afin d'avancer régulièrement.
- **Et n'hésitez pas à poser des questions !**

Exercice 6 - A vous de jouer !

- Comparez la grammaire du sujet avec le code fourni.
- Écrivez la fonction `ast_file` (fichier `pretty.ml`) permettant de visualiser vos premiers AST.
- Testez-la sur des exemples simples.
- Ajoutez les vérifications concernant "main" et "fmt".
- Réussir à typer les fichiers `hello-world.c`, `testfile-print-1.go` et `return.go`