INO + Geany

Ino + Geany

Ino

Telechargement

Ino est une boîte à outils en ligne de commande pour travailler avec le matériel Arduino

Cela vous permet de:

```
  Créez rapidement de nouveaux projets
  Construire un firmware à partir de plusieurs fichiers sources et
bibliothèques
  Télécharger le firmware sur un appareil
  Effectuer une communication série avec un périphérique (également appelé
moniteur série)
```

Ino peut remplacer l'interface utilisateur Arduino IDE si vous préférez utiliser la ligne de commande et un éditeur de votre choix ou si vous souhaitez intégrer le processus de création Arduino à l'IDE tiers.

Ino est basé sur make pour effectuer des builds. Cependant Makefiles sont générés automatiquement et vous ne les verrez jamais si vous ne le souhaitez pas. Caractéristiques

```
  Simple. Aucun script de build n'est nécessaire.
  Constructions hors-source. Les répertoires contenant des fichiers source
ne sont pas encombrés de fichiers objets intermédiaires.
  Prise en charge des esquisses * .ino et * .pde ainsi que des * .c et *
.cpp bruts.
  Prise en charge des versions 1.x du logiciel Arduino ainsi que celles de
0.x.
  Suivi automatique des dépendances Les bibliothèques référencées sont
automatiquement incluses dans le processus de construction. Les
modifications apportées aux fichiers * .h entraînent la recompilation des
sources qui les incluent.
  Sortie assez colorée.
  Prise en charge de toutes les cartes prises en charge par Arduino IDE.
  Vite. Les chemins d'outils découverts et d'autres éléments sont mis en
cache dans les exécutions. Si rien n'a changé, rien n'est construit.
  Flexible. Prise en charge des fichiers de configuration simples de style
ini pour configurer des informations spécifiques à la machine, comme le
modèle Arduino utilisé, le chemin de distribution Arduino, etc. juste une
fois.
```

Installation

De la source:

```
  Télécharger la dernière archive tar source
  Ou copiez-le depuis GitHub: git clone git: //github.com/amperka/ino.git
  Effectuez l' installation pour effectuer l'installation sous / usr / local
  Ou voyez INSTALL pour des instructions sur le changement de répertoire de
destination
```

Avec les outils de configuration Python:

```
  Les deux pip installent ino
  Ou easy_install ino
```

Exigences

```
  Python 2.6+
  Distribution IDE Arduino
  picocom pour la communication série
```

Limitations

```
  Comme pour la version actuelle, ino ne fonctionne que sous Linux et MacOS.
Cependant, il a été créé en pensant aux autres utilisateurs du système
d'exploitation, de sorte qu'il obtiendra éventuellement un support
multiplateforme complet. L'aide des développeurs Windows est très appréciée.
```

Obtenir de l'aide

```
  Jetez un oeil à tutoriel de démarrage rapide .
  Exécutez ino --help .
  Publier des problèmes à GitHub .
```

Licence

Si non indiqué autrement ino est distribué en termes de licence du logiciel MIT. Voir MIT-LICENSE.txt dans la distribution pour plus de détails. Contributeurs

```
  David Charbonnier .
  Jared Boone .
  Lars Englund .
  Alberto Ruiz .
  12qu
  Michael Sproul .
  Marc Plano-Lesay .
  Fabian Kreiser .
```

Changelog

0.3.6

```
      Correctif # 74, # 107, # 108: L'utilisation des types déclarés dans
les fichiers inclus est autorisée dans les définitions de fonction.
```

```
Auparavant, cela a conduit à: '<enum | struct | typedef>' non déclaré dans cette portée.
      Correctif n ° 105: Recherchez avrdude.conf dans / etc / avrdude pour être compatible avec Fedora.
      Correctif n ° 99: Vérification d'un projet existant avant de créer ou de créer des répertoires
      Correctif # 93, # 57, # 8: Les drapeaux de compilation et de lien personnalisés peuvent être passés comme arguments de compilation ino
      Correctif # 60, # 63: La commande d'outil de création personnalisée peut être passée en argument de génération ino
      Fix # 23, # 28: make est recherché dans les binaires IDE Arduino
      Correction n ° 88, n ° 103: Corriger l'analyse de la version pour certaines distributions qui la déforment
      Correction # 46: Prise en compte du numéro de build dans la chaîne de version
      Fix # 19, # 81, # 82: Les arguments de ligne de commande personnalisés pour picocom peuvent être passés lors de l'exécution ino série
```

0.3.5

```
      Correction # 62: Inclure MIT-LICENSE.txt dans l'archive.
```

0.3.4

```
      Fix # 44, # 45: Construire et télécharger pour Arduino Leonardo est entièrement pris en charge.
      Fix # 3, # 29: Construire des artefacts pour différents modèles de cartes et les distributions Arduino vont dans différents sous-répertoires de construction, donc vous ne devez pas lancer ino clean et reconstruire si vous passez à un autre modèle Arduino ou à une distribution logicielle.
      La version du jeu d'outils avr gcc fournie avec le logiciel Arduino est désormais toujours préférée à l'ensemble du système. Alors que les utilisateurs avec des versions de bord de logiciels (tels que Arch Linux) capables de produire des résultats attendus.
```

0.3.3

```
      Correctif n ° 16: Les esquisses * .ino et * .pde sont maintenant remplies avec des prototypes de fonctions tandis que le prétraitement s'effectue de la même manière qu'Erduino IDE, il est donc désormais possible d'utiliser des fonctions avant qu'elles ne soient déclarées ou définies.
```

0.3.2

```
      Correctif # 13: L'en-tête local #includes des fichiers d'esquisse ne mène plus à l'erreur 'No tel fichier ou répertoire'. GCC reçoit maintenant un chemin d'inclusion supplémentaire pointant vers l'origine de l'esquisse lors de la compilation de la source traitée.
      Correctif n ° 18: Analyse correcte des fichiers de dépendance lorsque plusieurs dépendances de bibliothèque sont trouvées sur la même ligne.
```

```
Maintenant, tous sont pris en compte, pas seulement le premier.
      Ajouter: Les fichiers d'esquisse traités ont maintenant la directive
#line afin qu'ils apparaissent comme source d'origine dans la sortie GCC en
cas d'erreurs de syntaxe.
      Ajouter: suivi automatique des dépendances pour les fichiers d'en-tête
inclus. Maintenant, une esquisse ou une source cpp se reconstruit une fois
qu'un en-tête inclus (directement ou indirectement) change.
```

0.3.1

```
      Support pour la construction ino --verbose
```

0.3.0

```
      Prise en charge de MacOS
      Estimation du port série
```

0.2.0

```
      Prise en charge de la version 1.0 du logiciel Arduino
```

0.1.x

```
      Première version
      Diverses corrections de bugs
```


Ino Demmarrage rapide

Démarrage rapide

Apprenez à travailler avec ino en quelques minutes. Créer un projet

Créons un projet simple:

```
$ mkdir bip
```

$ cd bip $ ino init -t clignote

Ici, nous avons créé un nouveau répertoire pour notre projet et initialisé le squelette du projet avec la commande ino init . Nous avons choisi d'utiliser blink comme modèle de projet. Cela va créer une simple esquisse pour le clignotement des LED sur la broche 13. Voyons ce que nous avons:

```
$ arbre
```

. ├── lib └── src

```
   └── sketch.ino
```

$ cat src / sketch.ino

```
#define LED_PIN 13

void setup() {
   pinMode (LED_PIN, OUTPUT);
}

Boucle vide () {
   digitalWrite (LED_PIN, HIGH);
   retard (100);
   digitalWrite (LED_PIN, LOW);
   retard (900);
}
```

Ici nous avons deux répertoires. src est un répertoire source où l'on peut placer les fichiers sources *. [c | cpp | pde | h | hpp] de notre projet. sketch.ino a été créé pour nous, nous avons donc un point de départ. lib est un répertoire où nous pouvons mettre des bibliothèques tierces si nous le voulons. Bâtiment

Construisons-le:

```
$ ino build
```

Recherche de fichier de description du forum (boards.txt) ... /usr/local/share/arduino/hardware/arduino/boards.txt Recherche de la bibliothèque principale Arduino ... / usr / local / share / arduino / matériel / arduino / core / arduino Recherche de bibliothèques standard Arduino ... / usr / local / share / arduino / libraries Recherche du fichier de version d'Arduino lib (version.txt) ... /usr/local/share/arduino/lib/version.txt Détection de la version du logiciel Arduino ... 22 Recherche d'avr-gcc ... / usr / bin / avr-gcc Recherche d'avr-g ++ ... / usr / bin / avr-g ++ Recherche d'avr-ar ... / usr / bin / avr-ar Recherche d'avr-objcopy ... / usr / bin / avr-objcopy Analyse des dépendances de src src / sketch.cpp arduino / wiring_shift.c arduino / wiring.c arduino / WInterrupts.c arduino / wiring_digital.c arduino / wiring_pulse.c arduino / wiring_analog.c arduino / pins_arduino.c arduino / HardwareSerial.cpp arduino / WString.cpp arduino / main.cpp arduino / Print.cpp arduino / WMath.cpp arduino / Tone.cpp Lier libcore.a Lier le firmware.elf Conversion en firmware.hex

Ouf! Beaucoup de travail a été fait en coulisses pour une seule commande. Il s'agit tout au plus de trouver les outils et les répertoires nécessaires et de compiler la bibliothèque principale standard. En fait, tu ne devrais pas t'inquiéter. La conséquence est que nous avons firmware.hex prêt à télécharger le fichier binaire. Téléchargement

Permet de le télécharger:

```
$ ino télécharger
```

À la recherche de stty ... / bin / stty Rechercher avrdude ... / usr / local / share / arduino / matériel /

tools / avrdude Rechercher avrdude.conf ... /usr/local/share/arduino/hardware/tools/avrdude.conf

avrdude: périphérique AVR initialisé et prêt à accepter les instructions

Lecture | ## | 100% 0.00s

avrdude: Signature de l'appareil = 0x1e950f avrdude: lecture du fichier d'entrée ".build / firmware.hex" avrdude: écriture flash (428 octets):

Rédaction | ## | 100% 0,08s

avrdude: 428 octets de flash écrits avrdude: vérification de la mémoire flash sur .build / firmware.hex: avrdude: charge les données de données flash du fichier d'entrée .build / firmware.hex: avrdude: le fichier d'entrée .build / firmware.hex contient 428 octets avrdude: lecture des données flash sur puce:

Lecture | ## | 100% 0,06s

avrdude: vérification ... avrdude: 428 octets de flash vérifiés

avrdude: safemode: Fusibles OK

avrdude fait. Je vous remercie.

Encore une fois, beaucoup de sortie, mais le travail est terminé. Arduino clignote avec sa LED intégrée sur la broche 13. Communication série

OK, maintenant traitons un peu avec la communication série. Avec l'éditeur de votre choix, changez src / sketch.ino en:

```
void setup()
```

{

```
   Serial.begin (9600);
```

}

Boucle vide () {

```
   Serial.println (millis ());
   retard (1000);
```

}

Cela devrait transmettre le nombre de millisecondes passées à partir de la mise sous tension chaque seconde via un port série. Construisons-le:

```
$ ino build
```

Analyse des dépendances de src src / sketch.cpp Lier le firmware.elf Conversion en firmware.hex

Comme vous pouvez le voir beaucoup moins de mesures ont été effectuées dans les coulisses. C'est parce que seules les choses qui ont été changées sont prises en compte. Cela booste la construction.

Permet de le télécharger avec le téléchargement ino . Lorsque le téléchargement est terminé, permet de se connecter à l'appareil avec un moniteur série pour voir ce qu'il imprime:

```
$ ino série
```

Recherche de moniteur série (picocom) ... / usr / bin / picocom picocom v1.4

le port est: / dev / ttyACM0 flowcontrol: aucun le débit en bauds est: 9600 la parité est: none les databits sont: 8 échapper est: Ca noinit est: non noreset est: non nolock est: oui send_cmd est: ascii_xfr -s -v -l10 receive_cmd est: rz -vv

Terminal prêt 0 1000 2004 3009 4014

C'est ce que nous voulons! Appuyez sur Ctrl + A Ctrl + X pour quitter. Ajuster les paramètres

Tous les exemples ont été faits en supposant que vous avez Arduino Uno et il est disponible sur le port par défaut. Maintenant, considérons que vous avez Arduino Mega 2560 et il est disponible sur le port / dev / ttyACM1 . Nous devons spécifier cela pour nos étapes de construction en tant que commutateurs de ligne de commande.

Le modèle de carte peut être réglé avec l' option –board-model ou -m . Le port est défini avec l' option –serial-port ou -p . Alors faisons-le:

```
$ ino build -m mega2560
```

$ ino télécharger -m mega2560 -p / dev / ttyACM1 $ ino série -p / dev / ttyACM1

Pour la liste complète des noms de cartes, reportez-vous à ino build –help . Fichiers de configuration

Il peut être ennuyeux de fournir ces commutateurs encore et encore. Vous pouvez donc les enregistrer dans le fichier ino.ini du répertoire du projet. Mettez les lignes suivantes à l' ino.ini :

```
[construire]
```

board-model = méga2560

[télécharger] board-model = méga2560 serial-port = / dev / ttyACM1

[en série] serial-port = / dev / ttyACM1

Maintenant, vous pouvez construire, télécharger et communiquer via la série sans avoir à fournir de paramètres. Eh bien, dans la plupart des cas, si vous construisez pour Mega 2560, vous aurez envie de télécharger sur Mega 2560 ainsi. La même chose à propos du réglage du port série. Donc, pour ne pas répéter les réglages pour différentes commandes, les commutateurs partagés pourraient être déplacés vers une section non nommée. Avoir juste les lignes suivantes dans ino.ini est suffisant:

```
board-model = méga2560
```

serial-port = / dev / ttyACM1

En outre, si vous avez Mega 2560, il est probable que vous l'ayez pour tous les projets que vous faites. Vous pouvez mettre un fichier de configuration partagé à:

```
/etc/ino.ini
~ / .inorc
```

Et il sera utilisé pour définir les valeurs de paramètres par défaut si elles ne sont pas remplacées par le ino.ini local ou par des commutateurs de ligne de commande explicites.

Vous pouvez fournir tous les arguments que vous utilisez sur la ligne de commande dans un fichier de configuration. Juste spécifier son nom long sans meneur - . Par exemple arduino-dist mais pas –arduino-dist ou -d . © Copyright 2011, équipe Amperka. Créé en utilisant Sphinx 1.1.3.

Geany

Présentation de l'IDE Geany

Geany est assez léger et comporte peu de dépendances. Il possède la coloration syntaxique et supporte pas mal de types de fichiers. Citons le HTML, XML, CSS, LaTeX, **C, C++,** PHP, **Python**, Perl, Lua, **Arduino**, ... Il permet aussi de lister les fonctions utilisées. Et, cerise sur le gâteau, le terminal est intégré, pas besoin de changer de fenêtre pour compiler.