

GitWizard

Beschreibung:

Das Projekt GitWizard ist eine Art Plugin um Git-Befehle aus der Lazarus IDE heraus ausführen zu können.

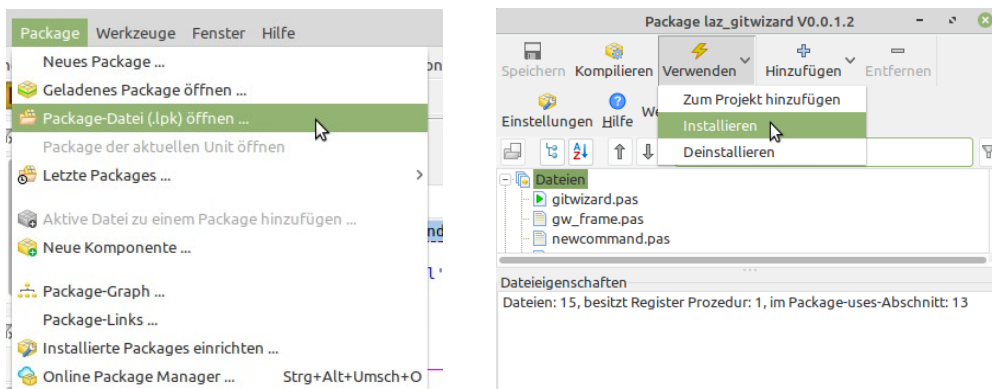
Der Grundgedanke des Programms ist, dass der Lazarus-Benutzer kleine Skriptdateien mit Git-Befehlen erstellen und diese dann mit GitWizard ausführen kann.

Installation:

Nachdem GitWizard von hier:

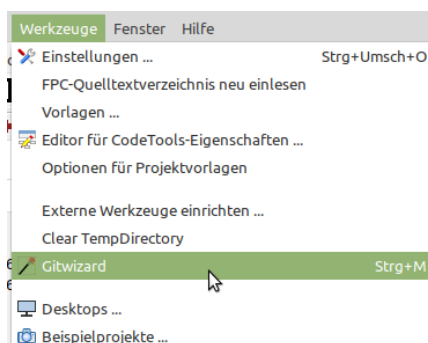
<https://github.com/wennerer/Gitwizard.git>

herunter geladen wurde, navigiert man in Lazarus zur laz_gitwizard.lpk und installiert das Package.

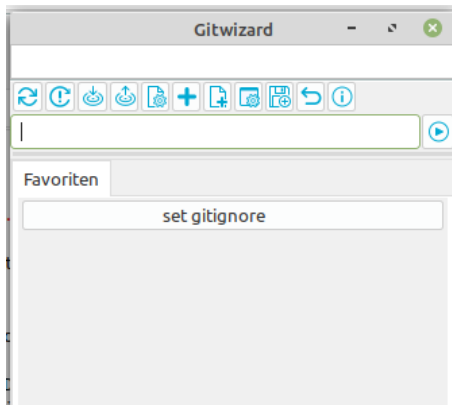


Einstellungen

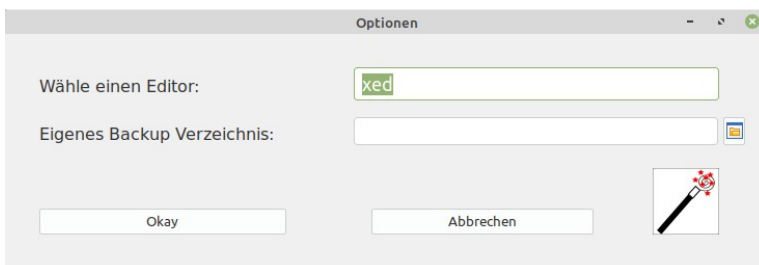
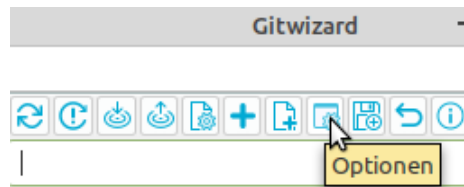
Sobald GitWizard erfolgreich installiert wurde befindet sich im Menü Werkzeuge ein neuer Eintrag.



Klickt man auf diesen neuen Eintrag öffnet sich der GitWizard.

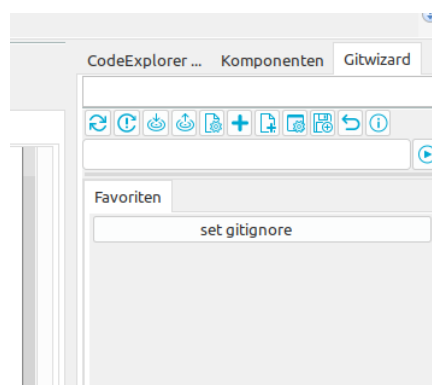
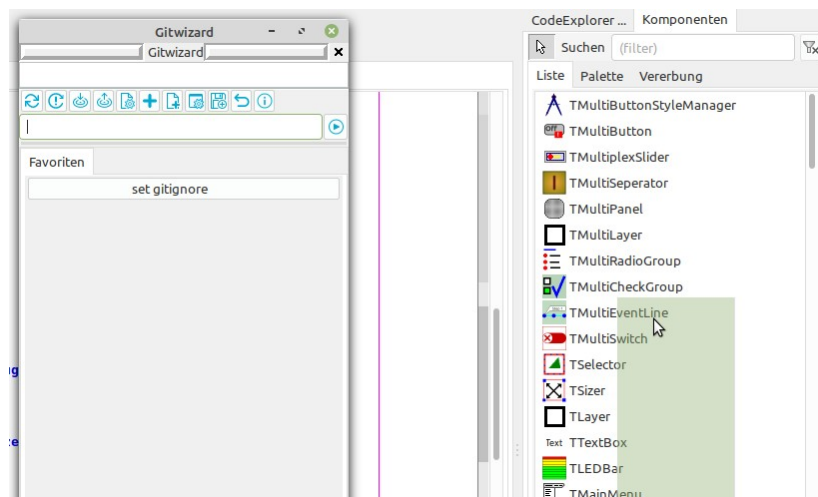


Bevor Sie anfangen mit ihm zu arbeiten klicken Sie bitte auf den Button „Optionen“ und geben einen Editor ihrer Wahl ein. Mit diesem werden dann die Skript-Dateien oder gitignore geöffnet.



Bei eigenes Backup Verzeichnis kann man das Verzeichnis hinterlegen in welches man seine Befehle sichern möchte. Es wird dann als Initialverzeichnis beim Öffnen der Backup-Dialoge benutzt.

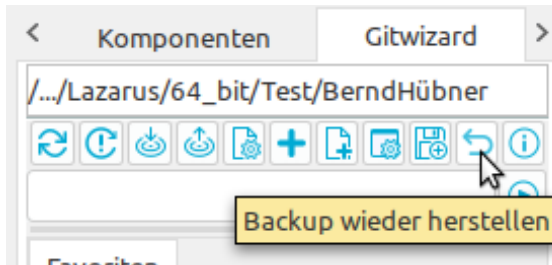
Möchten Sie GitWizard in der IDE andocken so funktioniert das wie bei allen anderen Fenstern auch. Machen Sie die header sichtbar und ziehen Sie das Fenster an die gewünschte Position.



Erste Schritte

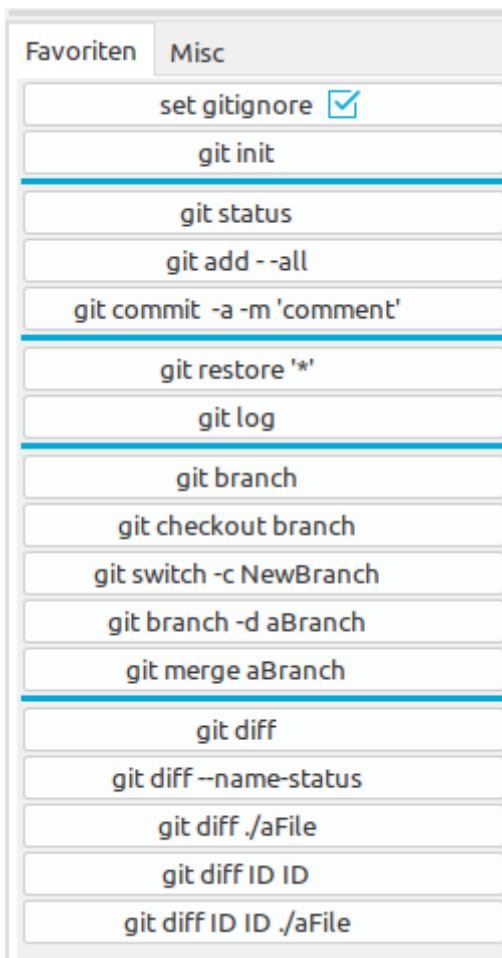
Bereitgestellte Befehle nutzen

Möchte man die mitgelieferten Befehle nutzen um den Umgang mit GitWizard kennen zu lernen kann man den Button „Backup wieder herstellen“ nutzen.



Die erste Abfrage mit ja beantworten, dann die Befehle aus dem Verzeichnis providedCommands/linuxCommands bzw. providedCommands\winCommands laden. Nun befindet sich eine Auswahl an Befehlen im GitWizard.

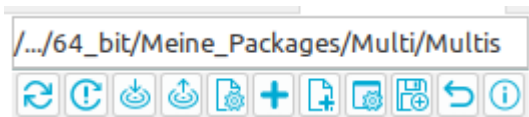
ACHTUNG: Der zuvor gewählte Editor wird dabei mit den von mir verwendeten Editoren xed bzw. notepad überschrieben. Also bitte gegebenenfalls nochmal ändern!!!



Es wurde auch ein zusätzlicher Tab erstellt auf dem ebenfalls Befehle sind.

Tipp: Möchte man danach alles leeren und seine eigenen Befehle erzeugen muss man im Lazarus Config-Ordner die Dateien gw_commands.xml und gw_options.xml löschen. Ebenfalls sollte man den Ordner linuxCommands bzw. winCommands leeren.

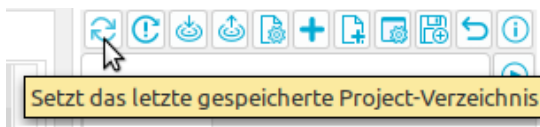
Verzeichnis-Panel



Im Verzeichnis-Panel ist ersichtlich welches Verzeichnis aktuell als Git-Projektverzeichnis gesetzt ist.

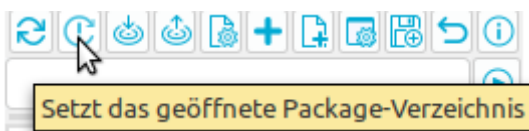
Die Toolbar Buttons

Der erste Button



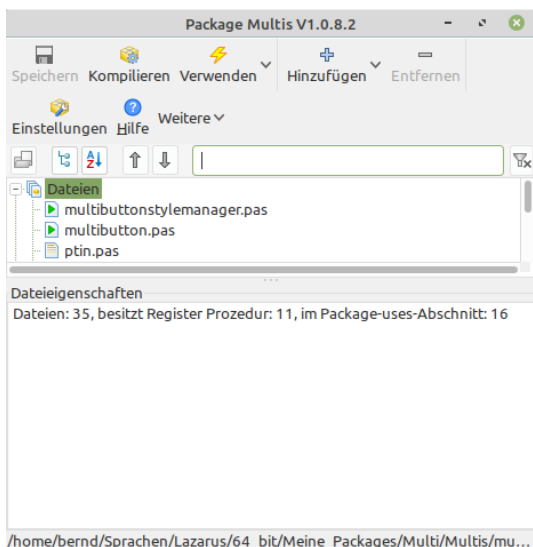
wird verwendet um das letzte in Lazarus gespeicherte Projekt als Git-Projektverzeichnis zu setzen.

Der zweite Button

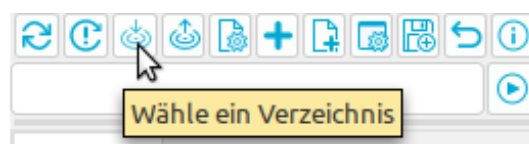


wird verwendet um ein geöffnetes Package als Git-Projektverzeichnis zu setzen.

Damit dies funktioniert muss ein Package Fenster wie dieses geöffnet sein. Aber Achtung, sind mehrere solcher Fenster geöffnet schlägt das Kommando fehl.

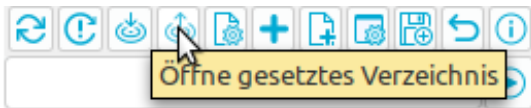


Der dritte Button



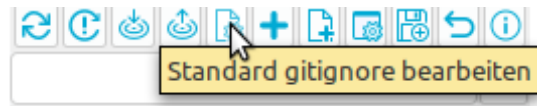
ermöglicht es mittels Dialog ein beliebiges Verzeichnis als Git-Projektverzeichnis zu setzen.

Mit dem vierten Button



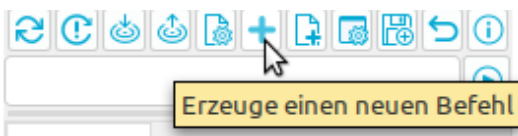
kann man das gesetzte Git-Projektverzeichnis im Standard-Explorer öffnen.

Fünfter Button



Im GitWizard Projektverzeichnis befindet sich eine Standard-Gitignore-Datei, welche mittels erstem Befehlsbutton in das Git-Projektverzeichnis kopiert werden kann. Mit dem fünften Toolbar-Button lässt sich diese Datei zum Bearbeiten öffnen. Achtung: es muss ein Editor unter Optionen gesetzt sein!

Sechster Button



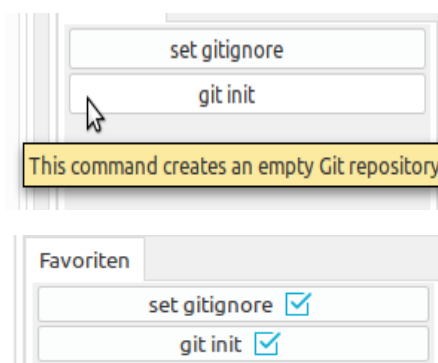
Mit dem sechsten Button lassen sich neue Befehle erzeugen. Dazu zwei Beispiele. Einmal ein Befehl bei dem keine weitere Eingabe benötigt wird und einmal ein Befehl bei dem bei Ausführung eine Eingabe nötig ist.

Einen einfachen Befehl anlegen:

Drückt man den sechsten Button (+) wird der Dialog zum Erzeugen eines neuen Befehls geöffnet:

A screenshot of the "Neuer Befehl Dialog" (New Command Dialog). It contains four input fields: "Bitte gib eine Caption für den neuen Button ein:" with "git init", "Bitte gib einen Hint für den neuen Button ein:" with "This command creates an empty Git repository", "Bitte gib einen Dateinamen für die Bashdatei ein:" with "git_init", and "Bitte gib einen neuen Befehl ein:" with "git init". At the bottom, there is a checkbox "Der Befehl benötigt eine Eingabe" which is unchecked, and two buttons: "Okay" and "Abbrechen".

Hier wird zum Beispiel der Befehl git init angelegt. Nachdem alles eingetragen wurde mit Okay bestätigen. Das Ergebnis sollte so aussehen:



Der Befehls-Button set gitignore ist immer vorhanden und kann auch nicht gelöscht werden. GitWizard erkennt ob im gesetzte Git-Projektverzeichnis eine .gitignore Datei und eine .git Datei vorhanden sind und zeigt dies mit einem blauen Haken an.

Einen Befehl mit Eingabe anlegen:

Drückt man den sechsten Button (+) wird der Dialog zum Erzeugen eines neuen Befehls geöffnet:

The 'Neuer Befehl Dialog' window contains the following fields and controls:

- Field: 'Bitte gib eine Caption für den neuen Button ein:' with value 'git commit -a -m 'comment''
- Field: 'Bitte gib einen Hint für den neuen Button ein:' with value 'Makes a commit and create a message for the commi'
- Field: 'Bitte gib einen Dateinamen für die Bashdatei ein:' with value 'git_commit_am'
- Field: 'Bitte gib einen neuen Befehl ein:' with value 'git commit -a -m '<comment>''
- Icon: A red star with a black pencil.
- Checkbox: 'Der Befehl benötigt eine Eingabe' (checked).
- Buttons: 'Okay' and 'Abbrechen'.

Hier wird der Befehl `git commit -a -m 'comment'` angelegt. Beim Ausführen des Befehls muss `comment` mit dem gewünschten Kommentar ersetzt werden. Damit dies möglich ist muss im Dialog der Haken bei „Der Befehl benötigt eine Eingabe“ gesetzt werden. Tipp: setzt man den zu editierenden Teil in `< >` wird dieser beim Aufruf selektiert.

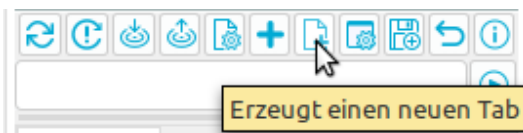
Es sollte dann so aussehen:

The 'Vervollständige den Befehl Dialog' window contains the following elements:

- Field: 'Vervollständige den Befehl:' with value 'git commit -a -m '<aComment>''
- Icon: A red star with a black pencil.
- Dropdown menu: 'Vordefinierte Argumente'.
- Buttons: 'Okay' and 'Abbrechen'.

Tipp: Mit Doppelklick auf Vordefinierte Argumente lassen sich neue Argumente hinzufügen.

Mit dem siebenten Button



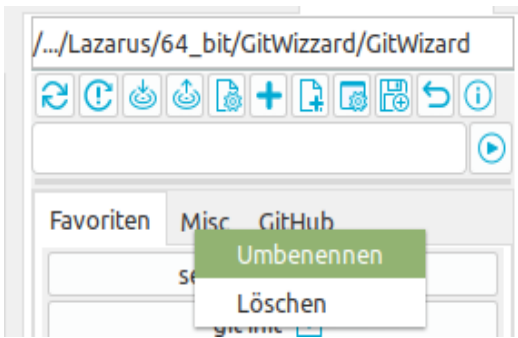
wird ein zusätzlicher Tab erstellt. Hat man nach einiger Zeit eine Reihe von Befehlen erzeugt kann es sinnvoll sein diese auf verschiedene Tab's zu verteilen.

The 'Create a NewTab' window contains the following elements:

- Icon: A red star with a black pencil.
- Text: 'Enter a caption:'
- Field: A text input field containing 'Misc'.
- Buttons: 'Okay' and 'Cancel'.

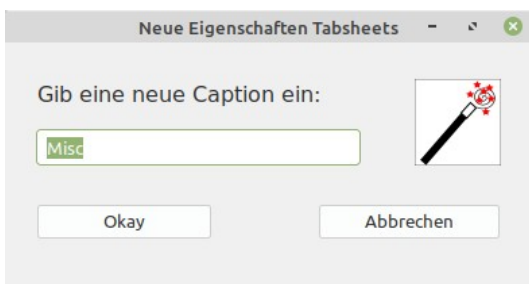
Tab's bearbeiten

Tipp: Klickt man mit der rechten Maustaste auf ein Tab erscheint ein Popup-Menü:



Achtung der Tab Favoriten lässt sich nicht bearbeiten!

Klickt man auf umbenennen kann man in einem Dialog eine neue Caption eingeben:

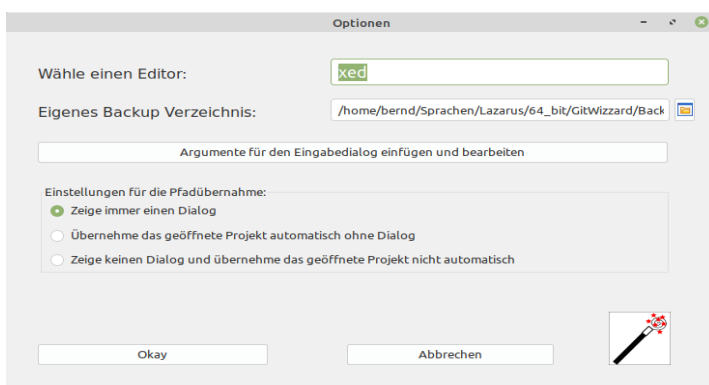


Löschen lässt sich ein Tab nur wenn sich keine Befehlsbuttons und/oder Seperatoren darauf befinden. Es müssen deshalb alle Befehlsbuttons und/oder Seperatoren in einen anderen Tab verschoben oder aber gelöscht werden.
Ist der Tab leer reicht ein Klick auf Löschen.

Der achte Button

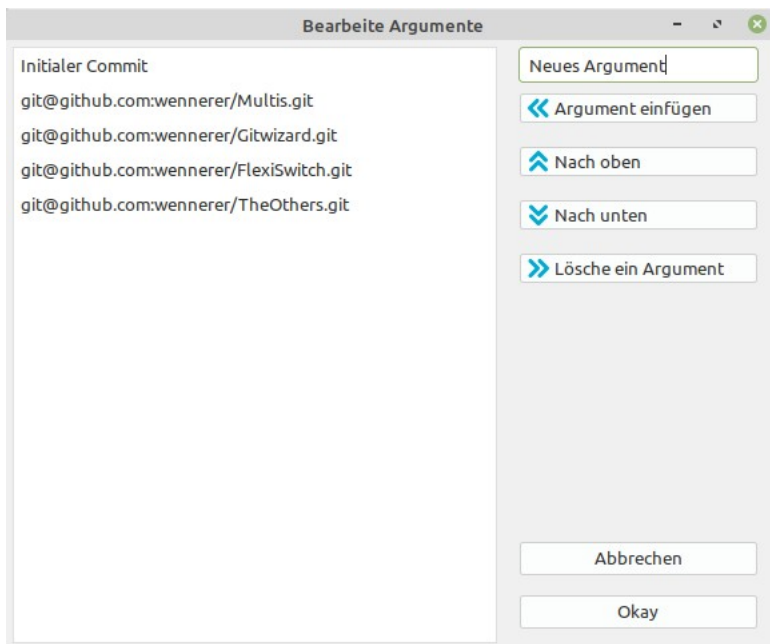


öffnet den Optionen Dialog. Hier muss ein Editor zum Öffnen der Skript-Dateien bzw. der gitignore Datei gewählt werden.



Zum Beispiel: xed, gedit, notpad etc.
Bei eigenes Backup Verzeichnis kann man das Verzeichnis hinterlegen in welches man seine Befehle sichern möchte. Es wird dann als Initialverzeichnis beim Öffnen der Backup-Dialoge benutzt.

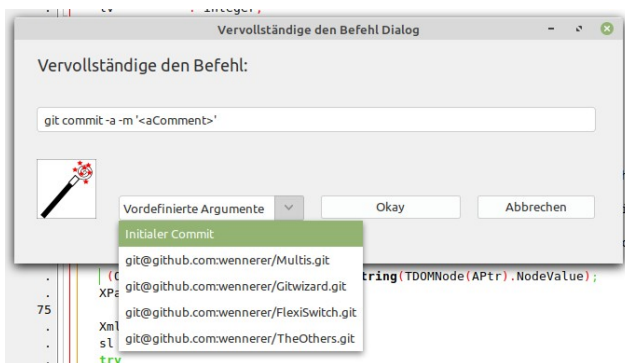
Klickt man auf Argumente für den Eingabedialog einfügen und bearbeiten dann öffnet sich folgendes Fenster:



Hier können Argumente hinterlegt werden die man dann im Befehlseingabefenster auswählen kann. Im Editfeld werden neue Argumente eingegeben und mit Argument einfügen hinzugefügt.

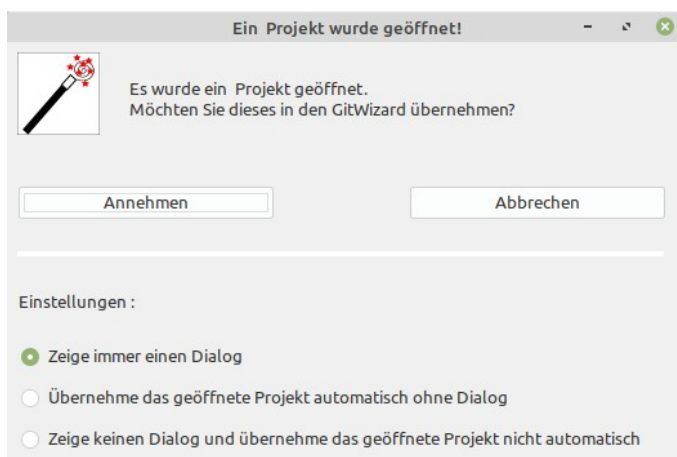
Mit Nach oben bzw. Nach unten kann die Reihenfolge geändert werden.

Lösche ein Argument entfernt dieses aus der Liste.



Einstellungen für die Pfadübernahme

Im Optionen-Dialog hat man Möglichkeiten zu Regeln wie der Pfad des Projektes beim Öffnen eines bestehenden Projektes in den GitWizard automatisch übernommen werden kann. Wählt man mit Dialog erfolgt eine Abfrage sobald ein bestehendes Projekt geöffnet wird.



In diesem Dialog besteht ebenfalls die Möglichkeit eine Auswahl zu treffen.

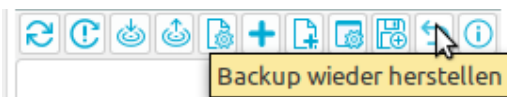
Wichtig! Wird ein neues Projekt angelegt wird der Pfad **nicht** automatisch übernommen.

Der neunte Button



öffnet einen Verzeichnis-Dialog. Als Initialverzeichnis istGitWizard/providedCommands/linuxCommands bzw. winCommands eingestellt. In diesem Verzeichnis befinden sich die mitgelieferten Befehle. Es empfiehlt sich für ein eigenes Backup einen separaten Ordner zu wählen. Beim einem eventuellen Update von GitWizard kommt man so nicht in Konflikte. Achtung: Macht man ein Backup der eigenen Befehle wird der Inhalt des Backup-Verzeichnisses erst gelöscht und dann werden die neuen Befehle reinkopiert!

Der zehnte Button

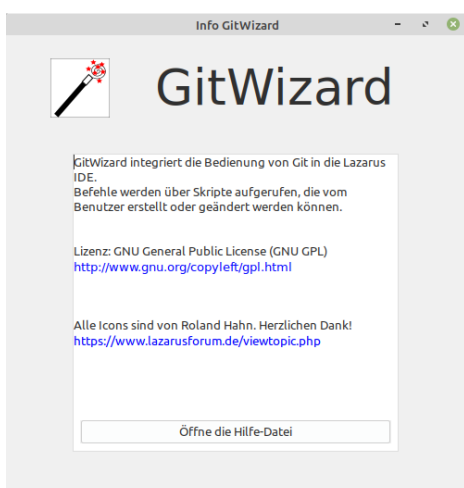


öffnet einen Verzeichnis-Dialog. Als Initialverzeichnis istGitWizard/providedCommands/linuxCommands bzw. winCommands eingestellt. In diesem Verzeichnis befinden sich die mitgelieferten Befehle. Achtung: Stellt man den zuletzt gesicherten Zustand wieder her, wird zuerst der Inhalt des OrdnersGitWizard/ linuxCommands bzw. winCommands gelöscht und dann die Backup-Dateien hineinkopiert.

Der letzte Button



öffnet ein kleines Info-Fenster.



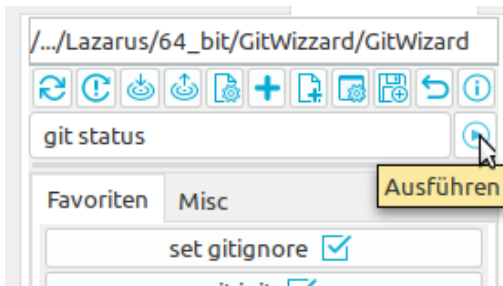
Klickt man den Button öffnet sich diese Hilfe-Datei.

Vielen Dank an Roland Hahn für die Images!

<https://www.lazarusforum.de/viewtopic.php?f=1&t=14263&p=128092&hilit=hahn#p128092>

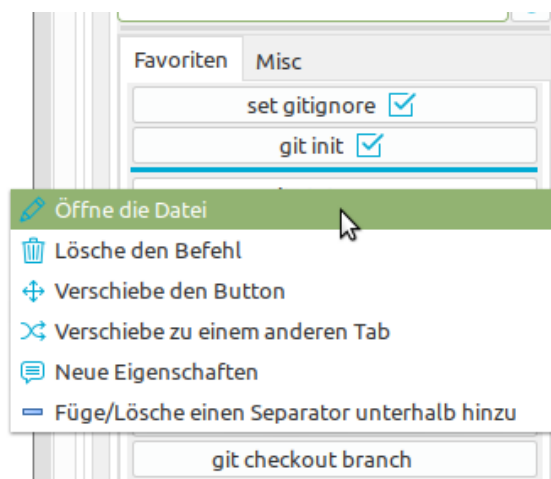
Einzelne Befehle ausführen

Um einzelne Befehle auszuführen (zu testen) gibt es eine Eingabezeile. Dort einfach den gewünschten Befehl eingeben (hier git status) und entweder Enter drücken oder den Ausführen-Button nutzen.



Befehl-Buttons bearbeiten

Alle Befehl-Buttons besitzen ein PopUp-Menü das sich durch Rechtsklick auf den Button öffnet.



Öffne die Datei, öffnet die Skript-Datei zum Bearbeiten.

Lösche den Befehl, löscht den Befehl aus der gw_commands.xml und die Skript-Datei aus dem Command-Ordner.

Verschiebe den Button, verschiebt den Button innerhalb des gleichen Tab's.

Verschiebe zu einem anderen Tab, verschiebt den Befehl zu einem anderen Tab.

Neue Eigenschaften, ermöglicht die Caption und den Hint zu ändern.

Füge/Lösche einen Separator unterhalb hinzu, fügt bzw. entfernt unterhalb des Befehlsbuttons einen Separator.

Das Ausgabefenster

```
1 diff --git a/gw_frame.pas b/gw_frame.pas
2 index df476a3..c696222 100755
3 --- a/gw_frame.pas
4 +++ b/gw_frame.pas
5 @@ -83,7 +83,7 @@ type
6     procedure movetotabClick({%H-}Sender: TObject);
7     procedure PageControl1Change(Sender: TObject);
8     procedure PageControl1MouseDown({%H-}Sender: TObject; Button: TMouseButton;
9         {%H-}Shift: TShiftState; X, Y: Integer);
10 +     procedure PageControl1MouseDown({%H-}Sender: TObject; Button: TMouseButton;
11         {%H-}Shift: TShiftState; X, Y: Integer);
12     procedure propertiesClick({%H-}Sender: TObject);
13     procedure ReadValues;
14     procedure renameClick({%H-}Sender: TObject);
15 diff --git a/gw_highlighter.pas b/gw_highlighter.pas
16 index b9bca03..94d2bdf 100644
17 --- a/gw_highlighter.pas
18 +++ b/gw_highlighter.pas
19 @@ -29,8 +29,6 @@ protected
20     fRange      : TRangeState; //definiert die Kategorien von Token
21     fHeaderCount : integer;
22 -    Durchlauf   : integer; //nur für Testzwecke debugln
23 -
24     fAtriSpace   : TSynHighlighterAttributes;
25     fAtriString  : TSynHighlighterAttributes;
26
27 @@ -128,8 +126,6 @@ begin
28     AddAttribute(fAtriSpace);
```

Lesezeichen:
 Setze ein Lesezeichen | Gehe zu Lesezeichen 0 | Lösche alle Lesezeichen | Lösche ein Lesezeichen
Falten:
 Alles Falten | Alles Entfalten
Suchen:
 Suche: frange
 schließen

Der Highlighter und die Faltmöglichkeiten im Ausgabefenster sind für ein git diff optimiert. Es ist möglich bis zu 10 Lesezeichen zusetzen und nach Zeichen zu suchen.