# Tcl/Tk

**Początki Tcl / Tk**

Tcl oznacza Tool Command Language. Tk jest rozszerzeniem graficznym Toolkit Tcl, udostępniającym różne standardowe elementy interfejsu GUI, które ułatwiają szybkie tworzenie aplikacji na wysokim poziomie. Opracowanie na Tcl / Tk, wymawiane "tickle tee-kay", rozpoczęło się w 1988 r. Od Johna K. Ousterhouta profesora U.C. Berkeley (UCB). Tcl został zaprojektowany z konkretnymi celami rozciągliwości, płytkiej krzywej uczenia się i łatwości osadzania. Rozwój TK rozpoczął się w 1989 roku, a pierwsza wersja była dostępna w 1991 roku. Dr Ousterhout kontynuował rozwój Tcl / Tk po opuszczeniu UCB, a następnie pracował dla Sun Microsystems. Teraz w Tcl Developer Xchange (która jest początkiem Ajuba Solutions, która została zakupiona przez Interwoven, kontynuuje ulepszanie języka, obecnie w wersji 8.3.2, stabilnej i wersji 8.4.

**Narzędzia i pliki**

Istnieją dwa główne programy, których potrzebujesz w systemie Linux do eksploracji Tcl / Tk. To są tclsh i wish. Jak można się domyślić od jego nazwy, pierwsza jest powłoką Tcl, najczęściej używaną do zapewnienia kontekstu wykonania skryptu powłoki. whish jest odpowiednikiem dla okienkowego środowiska graficznego. Sprawdź obecność tych plików, wpisując:

~/tcltk$ which tclsh

/usr/bin/tclsh

~/tcltk$ which wish

/usr/bin/wish

Komenda która zwraca ścieżkę do określonego pliku wykonywalnego. Jeśli nie widzisz wyników podobnych do tych, powinieneś udać się na stronę Tcl / Tk, aby pobrać i zbudować własną kopię. Oczywiście brak tych programów w twoim systemie nie wskazuje na żaden problem. W przeciwieństwie do Perla, Tcl / Tk zasadniczo nie jest uważany za niezbędny do działania systemu Linux. Każda dystrybucja, o której wiem, że jest z wersją Tcl / Tk i najpopularniejszymi rozszerzeniami Jeśli masz trudności, skontaktuj się z wydawcą oprogramowania GNU / Linux.

**Język Tcl**

Co wyróżnia Tcl?

Poniżej mamy wspólny pierwszy przykładowy program, zaimplementowany w Tcl. Jest to kompletny skrypt: pierwsza linia wywołuje środowisko tclsh, a druga wykonuje rzeczywistą pracę. Utwórz skrypt za pomocą wybranego edytora tekstu, ustaw go jako wykonywalny, wpisując chmod + x hello.tcl, a następnie wykonaj go, aby przetestować swoje dzieło.

~/tcltk$ cat hello.tcl

#!/usr/bin/tclsh

puts stdout {Hello, World!}

~/tcltk$ ./hello.tcl

Hello, World!

Tcl i Tk są interpretowanymi, rozszerzalnymi językami skryptowymi. Licencja, która jest bardzo podobna do licencji BSD, zezwala na swobodne korzystanie w każdych okolicznościach, o ile prawa autorskie są zachowywane we wszystkich kopiach, a powiadomienia są przekazywane dosłownie w każdej dystrybucji. Warunki licencji tworzą wolne oprogramowanie Tcl / Tk. Tcl / Tk jest środowiskiem interpretowanym. Interpreter Tcl można rozszerzyć, dodając wstępnie skompilowane funkcje C, które można wywoływać z poziomu środowiska Tcl. Rozszerzenia te mogą być niestandardowe w określonym celu lub ogólne i powszechnie przydatne. Przyjrzymy się kilku rozszerzeniom w dalszej części samouczka, ze szczególnym uwzględnieniem pierwszego rozszerzenia, bardzo popularnego Expect. W następnych kilku panelach przejrzymy główne cechy języka Tcl, od metaznaków i zmiennych globalnych, po operatorów, funkcje matematyczne i podstawowe polecenia. W końcu polecenia sprawiają, że Tcl / Tk jest charakterystycznym ewoluującym językiem. Pamiętaj, że w tym samouczku nie ma miejsca na pokrycie wszystkich poleceń.

**Metaznaki Tcl**

Metaznakami są te znaki lub pary znaków, które mają specjalne znaczenie w kontekście środowiska Tcl / Tk, w tym wyrażenia grupujące, enkapsulujące łańcuchy, wyciągi kończące i inne, jak zaznaczono w poniższej tabeli. Wiele z nich wykazano w poniższym kodzie. Jedną szczególną cechą, którą należy zauważyć, jest różnica w wynikach, gdy nawiasy klamrowe (które zapobiegają podstawianiu i rozszerzaniu) są używane zamiast podwójnych cudzysłowów.

```
#!/usr/bin/tclsh

# filename hello2.tcl

# This program code shows

# metacharacter usage

puts stdout "Hello, World! \a"

puts stdout {Hello, World! \a}

set Pints 6

set Days 7

puts stdout "The answer to the \

universe is \

[eval expr $Pints * $Days]!\n"
```

***

```
~/tcltk$ ./hello2.tcl

Hello, World!

Hello, World! \a

The answer to everything is 42!
```

Character(s) Used as

# Comment

; or newline Statement separators

Name A variable (case sensitive)

Name(idx) Array Variable

Name(j,k,l...) Multidimensional Array

"string" Quoting with substitution

{string} Quoting without substitution

[string] Command substitution

\char Backslash substitution

\ Line continuation (at end of line)

**Globalne zmienne Tcl i substytucje odwróconego ukośnika**

Istnieje kilka globalnych zmiennych (i są one predefiniowane, jeśli nie są puste w bieżącym kontekście), gdy zaczyna się skrypt Tcl / Tk. Te zmienne zezwalają na dostęp do środowiska operacyjnego w następujący sposób: argc to liczba argumentów skryptu, nie licząc nazwy jako wywoływanej. argv to lista (nie tablica) argumentów. argv0 to wywoływana nazwa pliku (która może być dowiązaniem symbolicznym). env to tablica, która jest indeksowana przez nazwy zmiennych środowiskowych obecnej powłoki. errorCode przechowuje informacje o najnowszym błędzie Tcl, a errorInfo zawiera ślad stosu z tego samego zdarzenia. Lista zawiera kolejne tuziny zmiennych tcl_xxx od tcl_interactive do tcl_version.

#!/usr/bin/tclsh

#

# Demonstrate global variables

# and backslash substitution

if {$argc >= 1} {

set N 1

foreach Arg $argv {

puts stdout "$N: $Arg\n"

set N [expr $N + 1]

if {$Arg == "ring"} {

puts stdout "\a"

}

}

} else {

puts stdout "$argv0 on \

X Display $env(DISPLAY)\n"

}

***

~/tcltk$ ./hello3.tcl

./hello3.tcl on X Display :0.0

~/tcltk$ ./hello3.tcl ring

1: ring

Kilka z tych zmiennych zostało użytych w powyższym przykładowym kodzie wraz z (jeszcze raz) niektórymi znakami ujętymi w ukośnik odwrotny, \ n i \ a. \ char umożliwia zamianę niedrukowalnych znaków ASCII. Jest to typowe dla wielu środowisk skryptowych i powłokowych w systemie UNIX. Jak wspomniano w tabeli, znak będący cytatem z ukośnikiem odwrotnym, który nie ma zdefiniowanego podstawienia, jest po prostu wysyłany do wyjścia.

\znak zastępowania

\a dzwonek

\b Backspace

\f znak ASCII kończący stronę

\n or \newline nowa linia

\r powrót karetki

\t tabulator poziomy

\v tabulator pionowy

\space ("\ ") spacja

\ddd wartość ósemkowa

\xddd... wartość szesnastkowa

\c Echo 'c'

\\ Backslash

**Operatory Tcl i funkcje matematyczne**

Tcl obsługuje standardową tablicę operatorów i funkcje matematyczne. Operatory obejmują operatory arytmetyczne, bitowe i logiczne, które są obliczane za pomocą polecenia expr, z wykorzystaniem zwykłych reguł priorytetu operatorów. Ponadto, biorąc pod uwagę fundamentalne korzenie Tcl jako zorientowanego na łańcuchy języka skryptowego, istnieje uzasadnione uzupełnienie funkcji matematycznych w następujący sposób:

* Funkcje trygonometryczne obejmują cos (x), acos (x), cosh (x), sin (x), asin (x), sinh (x), tan (x), atan (x), atan2 (y, x) , tanh (x) i hypot (x, y). Odpowiednią jednostką dla tych funkcji są radiany.

* Funkcje logarytmiczne to: exp (x), log (x) i log10 (x).

* Funkcje arytmetyczne to: ceil (x), floor (x), fmod (x, y), pow (x, y), abs (x), int (x), double (x) i round (x).

* Liczby losowe są obsługiwane przez rand () i srand (x).

Poniższy przykład wykorzystuje tylko kilka z tych operatorów i funkcji do obliczania odległości między określonym punktem a punktem początkowym oraz do zwracania obwodu i obszaru okręgu o określonym promieniu. Dodatkowo w tym przykładzie używamy komendy indeksu list (lindex), aby uzyskać dostęp do poszczególnych elementów $ argv.

```
~/tcltk$ cat maths.tcl

#!/usr/bin/tclsh

#

# Demonstrate operators and

# math functions

set PI [expr 2 * asin(1.0)]

if {$argc == 3} {

set X [lindex $argv 0]

set Y [lindex $argv 1]

set Rad [lindex $argv 2]

set Dist [expr sqrt(($X*$X)+($Y*$Y))]

set Cir [expr 2*$PI*$Rad]

set Area [expr $PI*$Rad*$Rad]

puts stdout "Distance = $Dist"

puts stdout "Circumference = $Cir"

puts stdout "Area = $Area"

} else {

puts stdout "Wrong argument count!"

puts stdout "Needs X, Y, and Radius"

}

********

~/tcltk$ ./maths.tcl 3 4 5

Distance = 5.0

Circumference = 31.4159265359

Area = 78.5398163397
```

**Pętle i rozgałęzienia w Tcl**

Polecenia pętli w Tcl to while, for i foreach. Warunkowe (rozgałęzione) polecenia to: / then / else / elsif i switch. Modyfikatory poprzednich poleceń to: break, continue, return i error. Na koniec komenda catch służy do obsługi błędów. jeśli / then / else / elsif zostało zademonstrowane w poprzednich panelach. Chociaż jest to część składni formalnej, najczęściej występuje w absencji. W poniższym przykładzie komenda switch jest podawana z argumentów wiersza poleceń przez konstruktor foreach. Gdy argumenty są przetwarzane (Uwaga: niewłaściwe dane wejściowe powodują zakończenie skryptu, ponieważ nie zaimplementowaliśmy przechwycenia błędu), pętla while przetwarza dane wejściowe, wywołując procedurę dla każdej linii i zwiększając licznik linii. Fragment kodu kończy się zwracając liczbę przetworzonych linii.

```
...
#
# parse command line switches
set Optimize 0
set Verbose 0
foreach Arg $argv {
switch -glob -- $Arg {
-o* {set Optimize 1}
-v* {set Verbose 1}
default {
error "Unknown $Arg"
}
}
}
set LineCount 0
while {[gets stdin Line] >= 0} {
# to confuse Vanna White...
Remove_Vowels $Line \
$Optimize $Verbose
incr LineCount
}
return LineCount
…
```

**Ciągi Tcl i dopasowywanie wzorców**

Ciągi są podstawowym typem danych w Tcl. Polecenie string jest naprawdę różnorodnym poleceniem, zebranym pod jednym parasolem.

```
~/tcltk$ tclsh
% set Phrase "hello, world!"
hello, world!
% string toupper $Phrase
HELLO, WORLD!
% string totitle $Phrase
Hello, world!
% string match ello $Phrase
0
% string match *ello* $Phrase
1
% string length $Phrase
14
% append Phrase "Nice day, eh?"
hello, world!
Nice day, eh?
% string toupper $Phrase
HELLO, WORLD!
NICE DAY, EH?
% string wordend $Phrase 7
12
```

Komendy łańcuchowe informacyjne to długość i długość bajtowa (które mogą się różnić w zależności od zestawu znaków). Porównania zwracające wartości logiczne (1 lub 0) to porównywanie, równość i dopasowanie. Dopasowywanie wzorca jest tutaj realizowane przez "globbing", prosty typ dopasowania zwykle kojarzony z operacjami powłoki. Zaawansowane wyrazy regularne są również dostępne za pośrednictwem odrębnych poleceń regex i regsub. Funkcje indeksowania w Tcl są wykonywane za pomocą poleceń index, last, first, wordend i wordstart. Modyfikacja ciągów jest obsługiwana przez tolower, toupper, totitle, trim, trimleft, trimright, replace i map. Ten ostatni wymaga wcześniejszego zdefiniowania tabeli odwzorowania znaków. Substringi są wyodrębniane z zakresu, a łańcuchy są wyprowadzane wiele razy z powtórzeniem. Tekst można dodać do istniejącej zmiennej za pomocą polecenia append. Polecenie format może być użyte do wygenerowania ciągów wyjściowych przy użyciu tych samych stylów i konwencji, co polecenie printf języka C. scan analizuje ciąg znaków i przypisuje wartości do zmiennych. W końcu, począwszy od Tcl 8.0, dodano funkcjonalność obsługi danych binarnych jako ciągów znaków (dzięki czemu można przetworzyć znak null bez awarii) za pomocą poleceń binarnych i poleceń skanowania binarnego.

Listy Tcl

Listy mają dwa główne zastosowania w Tcl. Pierwsze, które już widzieliśmy, demonstrowało w kontekście przetwarzania argumentów linii poleceń za pomocą polecenia foreach. Drugim zastosowaniem jest dynamiczne budowanie elementów polecenia Tcl, które można później wykonać za pomocą polecenia eval, jak widzimy w dalszej części tego samouczka. Polecenie list pobiera wszystkie swoje argumenty i zwraca je w kontekście listowym. Argumenty mogą być wartościami lub zmiennymi. Z poniższego przykładu listy można tworzyć ręcznie lub używając listy, która może przyjmować inne listy jako argumenty (w ten sposób zapisując orientację dwóch par naszej pierwszej "Strony"). Alternatywnie, polecenie concat służy do scalania dwóch lub więcej list w jedną całość elementów najwyższego poziomu, zwracając drugą, bardziej interesującą "Stronę".

~/tcltk$ tclsh

% set c1 {Bob Carol}

Bob Carol

% set c2 [list Ted Alice]

Ted Alice

% set Party1 [list $c1 $c2]

{Bob Carol} {Ted Alice}

% set Party2 [concat $c1 $c2]

Bob Carol Ted Alice

% linsert $Party1 1 Richard

{Bob Carol} Richard {Ted Alice}

%

Inne przydatne komendy list i ich składnia to:

* llength $ List - zwraca liczbę przedmiotów z toplevel.

* lindex $ Lista n - zwraca pozycję indeksowaną, liczy od zera.

* lrange $ List i j - zwraca zakres elementów listy.

* lappend $ Element listy ... - dołącz elementy do listy.

* linsert $ Lista n pozycji ... - wstawia element (y) w określonej pozycji na liście, przenosząc inne pozycje w dół listy.

Saldo poleceń listowych to: lreplace, lsearch i lsort. Komenda split pobiera ciąg znaków jako dane wejściowe i generuje poprawnie przeanalizowaną listę, rozbijając ciąg znaków pod określonym znakiem. join wykonuje operację komplementarną, biorąc elementy listy i łącząc je ze sobą, oddzielone łączeniem.

**Tablice Tcl**

Skrót do rozumienia tablic Tcl polega na traktowaniu ich w tym samym świetle, co w haśle Perla. Tablica nie jest indeksowaną numerycznie liniową strukturą danych, chyba że zdecydujesz się nałożyć taką interpretację na dane. Indeks (lub klucz) może być dowolnym ciągiem, chociaż ciągi ze spacjami muszą

być albo cytowane, albo odwołanie do zmiennej. Podobnie jak w przypadku zmiennych normalnych, tablice są inicjowane za pomocą polecenia set, jak pokazano poniżej. Indeks jest podany w nawiasach. Należy pamiętać, że nawiasy nie udostępniają grupowania, jak nawiasy klamrowe lub podwójne cudzysłowy. Po zainicjowaniu jako tablica, zmienna nie może być dostępna jako pojedyncza zmienna. Jak pokazano na dole listy poniżej, elementy tablicy mogą być również listami.

```
~/tcltk$ tclsh

% set People(friend) Tom

Tom

% set People(spouse) Marcia

Marcia

% set People(boss) Jack

Jack

% array names People

friend boss spouse

% set Person $People(friend)

Tom

% array get People

friend Tom boss Jack spouse Marcia

% set People(friend) \

[concat $People(friend) Bob]

Tom Bob

% set Person $People(friend)

Tom Bob

%
```

Więcej tablic Tcl

Polecenie tablica jest narzędziem wielofunkcyjnym, podobnie jak ciąg. Polecenia to tablica do testowania zmiennej dla istnienia jako tablica, tablica get do konwersji do formatu listy, zestaw tablic do konwersji z listy na tablicę, nazwy tablic do zwrócenia listy indeksów i rozmiaru tablicy, aby zwrócić liczbę indeksów . Przeszukiwanie tablicy ma swój własny zestaw czterech poleceń: tablica startseach, tablica już nie, następna tablica i tablica donesearch. Chociaż tablice Tcl są jednowymiarowe według projektu, istnieje elegancki sposób na symulowanie wielowymiarowych konstrukcji. Ponieważ indeksy są arbitralnymi łańcuchami, tablica 2D może być zadeklarowana w następujący sposób:

```
set i 1 ; set j 10

set array($i,$j) 3.14159

incr $j
```

set array($i,$j) 2.71828

Te klucze tablicowe to tak naprawdę tylko łańcuchy "1,10" i "1,11", ale dla celów dostępu do danych, kto ma wiedzieć różnicę?

**Procedury Tcl**

Polecenie proc definiuje procedurę Tcl. Po zdefiniowaniu procedura może zostać wywołana lub użyta tak, jak byłoby to wbudowane w polecenie Tcl. Dodatkowo można zdefiniować parametry za pomocą wartości domyślnych; na przykład zmiana poniższej definicji do odczytu proc c_area {{rad 1}} umożliwiłaby wywołanie procedury c_area bez parametrów, zwracając obszar koła jednostkowego.

```
#!/usr/bin/tclsh

#

# Demonstrate procedures and

# global scoping briefly

set PI [expr 2 * asin(1.0)]

proc circum {rad} {

global PI

return [expr 2.0 * $rad * $PI]

}

proc c_area {rad} {

global PI

return [expr $rad * $rad * $PI]

}

set rad 3

puts stdout "Area of circle of\

radius $rad is [c_area $rad],\n\

the circumference is\

[circum $rad].\n"
```

*********

```
~/tcltk$ ./protest.tcl

Area of circle of radius 3 is 28.2743338823,

the circumference is 18.8495559215.
```

Polecenie zmiany nazwy jest używane tak, jak brzmi, aby nadać nową nazwę istniejącemu poleceniu lub procedurze. Istnieją dwa wyraźne powody używania nazwy. Pierwszym jest dodanie funkcjonalności do istniejącego polecenia poprzez zmianę nazwy oryginału, a następnie zastąpienie go procedurą o tej samej nazwie. Procedura może wywoływać oryginał i dodawać potrzebne dzwonki i

gwizdki. Drugim powodem zmiany nazwy jest mapowanie polecenia z istnienia, na przykład zmiana nazwy exec {} ;, na przykład, aby uniemożliwić użytkownikom wykonywanie poleceń zewnętrznych.

**Reguły zasięgu zmiennych**

Reguły zakresu opisują widoczność procedury oraz nazwy zmiennych i wartości na różnych poziomach programu. Na przykład zmienne zdefiniowane na najbardziej zewnętrznym poziomie skryptu są zmiennymi globalnymi. Domyślnie są one niewidoczne, a ich wartości nie są dostępne w ramach procedur. Dzięki temu scenarzyści procedur mogą swobodnie definiować nazwy zmiennych i przypisywać wartości bez obawy o nadpisanie ważnych zmiennych nieznanych w lokalnym zasięgu. Aby zmienna globalna była widoczna wewnątrz procedury, musi zostać zadeklarowana jako taka w procedurze, tak jak zrobiłem to dla PI (w przykładzie na poprzednim panelu) przy użyciu polecenia globalnego. Polecenie upvar zapewnia możliwość powiązania zmiennej poziomu lokalnego z wartością zmiennej z innego zakresu. Pozwala to na wywoływanie według nazwy w procedurach, co jest przydatne, gdy procedura musi być w stanie zmienić wartość w innym zakresie, a nie tylko używać go. Składnia komendy to upvar level $ VarName LocalVar, gdzie level to liczba kroków wykraczających poza bieżący zakres. "# 0" jest reprezentacją globalnego poziomu zasięgu.

**Struktury danych w Tcl**

Poza prostymi wielowymiarowymi tablicami, ogólnie zaleca się, aby struktury danych Tcl były implementowane jako tablice, które mają dedykowane interfejsy proceduralne. Ten projekt ukrywa konkretne szczegóły implementacji od użytkownika struktur, zapewniając jednocześnie możliwość wykonywania istotnych funkcji sprawdzania błędów. W poniższym przykładzie, po zadeklarowaniu uData jako zmiennej globalnej, kod wykonuje kontrolę, aby zobaczyć, że konto jeszcze nie istnieje. Jeśli tak, procedura powraca z komunikatem o błędzie (niezerowym). Zwrot może być użyty w przełączniku, aby wygenerować wyjściowy tekst błędu. Na przykład podajemy po prostu trzy kolejne wejścia, w tym jedno powtórzenie. Daje to wynik pokazany na dole, przy czym "1" wskazuje na celowy powrót błędu z powodu powtarzającej się nazwy konta.

```
#!/usr/bin/tclsh

#

# Demonstrate Data Structures

# using procedural wrappers

proc UserAdd { Acct rName eMail phone } {

global uData

if {[info exists uData($Acct,rname)]} {

return 1

}

set uData($Acct,rname) $rName

set uData($Acct,email) $eMail

set uData($Acct,phone) $phone

return 0
```

```
}

puts stdout [UserAdd bpb\

Brian bilbrey@junk.com 555-1212]

puts stdout [UserAdd tom\

Tom tom@junk.com 555-1212]

puts stdout [UserAdd bpb\

Brian bilbrey@junk.com 555-1212]

******

~/tcltk$ ./datas.tcl

0

0

1
```

Inne możliwości struktur danych obejmują listy macierzy, powiązane lub podwójnie połączone macierze lub ich różne kombinacje. Listy konstrukcji tablic są znacznie wydajniejsze dzięki reimplementacji list, która towarzyszyła Tcl 8.0, zapewniając stały czas dostępu.

**Ścieżki i pliki**

Operacje na plikach i ścieżkach stanowią trudny problem w środowisku wieloplatformowym. Tcl używa nazw ścieżek UNIX (oddzielonych domyślnie znakiem "/"), a także macierzystej konstrukcji ścieżki dla hosta. Nawet jeśli dane w programie są poprawnie skonstruowane, może być trudno zapewnić, że dane wejściowe użytkownika odpowiadają wymaganiom systemu. Polecenie file join służy do konwersji formatów UNIX do natywnych ścieżek. Inne komendy łańcucha znaków obejmują podział pliku, nazwę, rozszerzenie pliku, nazwę własną, typ i ogon. W swojej roli "Tool Command Language", Tcl posiada szeroki zakres wewnętrznych testów plików i funkcji operacyjnych. Każde polecenie kończy się plikiem, tak jak w pliku istnieje nazwa. Inne komendy testowe (które zwracają wartości logiczne) zawierają pliki wykonywalne, katalogi, pliki isfile, własności, czytelności i możliwości zapisu. Informacje o pliku i operacje są wykonywane (ponownie, wszystkie z wiodącym plikiem) atime, atrybuty, kopiuj, usuń, lstat, mkdir, mtime, readlink, zmień nazwę, nazwę root, rozmiar, stat i typ. Należy zwrócić uwagę, że wiele poleceń dotyczących informacji o plikach może zwracać niezdefiniowane dane podczas działania w środowisku Windows lub Mac, ponieważ na przykład dane łącza i-węzła i symboliczne (i twarde) nie są reprezentowane w tych systemach plików. Zaletą używania poleceń file ... zamiast używania natywnych poleceń przez exec jest to, że pierwszy przedstawia przenośny interfejs. Cały ten dokument można z łatwością poświęcić tylko tej jednej sekcji języka Tcl.

```
~/tcltk$ tclsh

% file exists hello3.tcl

1

% file executable testit

0
```

% file pathtype ./hello3.tcl

relative

% set dir1 home

home

% set dir2 brian

brian

% set dir3 tcltk

tcltk

% file join /$dir1 dir2 dir3

/home/dir2/dir3

% file delete testit~

%

**Procesy i pliki I / O z Tcl**

Polecenie exec służy do jawnego wykonywania poleceń zewnętrznych. Pod Linuksem większość poleceń zewnętrznych można uruchomić bezpośrednio, gdy Tcl jest w trybie interaktywnym, jak pokazano w przykładzie poniżej. Uruchamianie z exec zwraca wyjście standardowe programu nie do ekranu, ale do Tcl, które pozwala na przypisanie danych do zmiennej. Gdy program jest uruchamiany w tle, natychmiastową wartością zwracaną jest PID dla programu. Programy exec'd mogą w pełni korzystać z przekierowania I / O i potoków w środowisku UNIX.

~/tcltk$ tclsh

% nslookup orbdesigns.com

Server: 192.168.1.3

Address: 192.168.1.3#53

Name: orbdesigns.com

Address: 64.81.69.163

% set d [date]

Sun Mar 25 13:51:59 PST 2001

% puts stdout $d

% set d [exec date]

Sun Mar 25 13:52:19 PST 2001

% puts stdout $d

Sun Mar 25 13:52:19 PST 2001

******

```
% if [catch {open foo r} Chan] {

puts stdout "Sorry, Dave...\n"

}

% gets $Chan

One

% gets $Chan

Two

% eof $Chan

0

% close $Chan

%
```

Inne komendy procesu to exit, który kończy działanie skryptu Tcl, i pid, który zwraca PID bieżącego (lub określonego) procesu, przydatny do różnych celów. Tcl nie zawiera żadnych natywnych poleceń sterujących procesem, ale można użyć polecenia exec w porozumieniu z danymi PID, aby wykonać wiele zadań. Manipulacja plikami wykorzystuje następujące polecenia: otwieranie, zamykanie, pobieranie, umieszczanie, czytanie, mówienie, wyszukiwanie, wysyłanie i opróżnianie. Jak pokazano powyżej, polecenie catch jest użyteczne podczas sprawdzania błędów podczas otwierania plików. Gdy dane wyjściowe programu muszą zostać wydrukowane przed napotkaniem znaku nowego wiersza, jak w monicie danych użytkownika, należy użyć opcji opróżniania, aby zapisać bufor wyjściowy. Dodatkową funkcją (w obsługiwanych środowiskach) jest możliwość otwierania potoków w taki sam sposób jak plik. Na przykład, po otwarciu kanału potoku z ustawionym Kanałem [otwórz "| sort foobar" r], wyjście pierwszego otrzyma będzie "Osiem" (alfabetycznie, poza danymi pliku "Jeden" do "Dziesięć", na 10 oddzielnych linii).

**Używanie eval do dynamicznego skryptowania**

W tym przykładzie możesz wyczuć moc polecenia eval. W normalnych warunkach interpreter Tcl działa w trybie jednoprzebiegowym: najpierw analizuje wejściową linię poleceń (ewentualnie rozciągniętą na kilka fizycznych linii), wykonując dowolne zmiany. Następnie wykonywane jest wykonanie, chyba że zostanie znalezione złe lub zniekształcone polecenie. eval pozwala na drugie przejście (lub może bardziej poprawnie, wstępne przejście). Zatem polecenie Tcl może być najpierw konstruowane dynamicznie, a następnie analizowane i wykonywane. W poniższym wykazie plik wejściowy składa się z trzech linii, z których każda ma jedną operację arytmetyczną w wierszu. Po wywołaniu tclsh plik jest otwierany tylko do odczytu i jest powiązany ze zmienną $ InFile. Pętla while czyta w jednym wierszu naraz, do $ Op. Następnie kompletne polecenie Tcl jest konstruowane przez wstępnie oczekiwany expr do zmiennej $ Op. Zostanie to następnie rozszerzone, ocenione i odpowiednio przypisany wynik. Na koniec każda operacja i wynik są wyświetlane na standardowym wyjściu.

```
~/tcltk$ cat input01.txt

1 + 2

4 + 5

7 - 9
```

```
~/tcltk$ tclsh

% set InFile [open input01.txt r]

file3

% while {[gets $InFile Op] >= 0} {

set Operation "expr $Op"

set Result [eval $Operation]

puts stdout "$Op = $Result\n"

}

1 + 2 = 3

4 + 5 = 9

7 - 9 = -2

%
```

Podczas gdy próbka ta pokazuje względnie banalne zastosowanie eval, koncepcyjnie można ją łatwo rozszerzyć na dynamiczne przetwarzanie plików i / lub katalogów w oparciu o dane wejściowe pliku wejściowego o znanej składni lub operacje bazujące na typie pliku, uprawnieniach, czasie dostępu lub dowolnym różnorodność testowalnych elementów.

**Komendy TK**

Czym w ogóle jest widget? Tk jest graficznym rozszerzeniem Toolkit dla Tcl. Wersje wydania TK są skoordynowane z wersjami Tcl. W następnych panelach przejrzymy zestaw widgetów TK, zbadamy niektóre opcje konfiguracji i skonfigurujemy kilka przykładów, aby zademonstrować użyteczną naturę Tk. Trudno jest przekonać dowolnego PHB (Pointy Haired Boss), że ta sekcja samouczka jest związana z pracą. W końcu chodzi o widżety, a widżety koncepcyjne są ściśle związane z grą ... ale to jest praca, więc zagłębimy się w nią. Po pierwsze, oto kod dla Tk wzbogacony "Hello, World!"

```
#!/usr/bin/wish

#

# Hello World, Tk-style

button .hello -text Hello \

-command {puts stdout \

"Hello, World!"}

button .goodbye -text Bye! \

-command {exit}

pack .hello -padx 60 -pady 5

pack .goodbye -padx 60 -pady 5
```

Wywołanie polecenia (powłoka TK) w pierwszym wierszu powoduje wyświetlenie widżetu okna o domyślnym rozmiarze. Następnie zdefiniowałem dwa widżety przycisków, .hello i .goodbye - są one pakowane do okna, a okno kurczy się do rozmiaru zdefiniowanego przez określony odstęp przycisków. Po uruchomieniu skryptu pojawia się okno dialogowe pokazane powyżej. Kliknij przycisk, aby uzyskać "Hello, World!" dane wyjściowe w oknie terminala nadrzędnego, Kliknij, aby zakończyć skrypt

**Widżety TK**

Istnieje bardzo niewiele poleceń używanych do tworzenia widżetów TK. Lepsza niż połowa to warianty przycisków lub widżetów tekstowych, jak widać na poniższej liście. Kilka z tych elementów zostało zademonstrowanych w następnym panelu.

* button - prosty widget z ponad dwudziestoma opcjami konfiguracji od kotwicy i czcionki po padx i relief.

* canvas - canvas to widget, który może zawierać nie tylko inne widżety, ale także zestaw uporządkowanych elementów graficznych, w tym okręgi, linie i wielokąty.

* checkbutton - tworzy widżet przycisku stylu checkbox, który jest powiązany ze zmienną.

* entry - buduje jednoliniowe pole wprowadzania tekstu.

* frame - ramka jest widżetem używanym głównie jako pojemnik lub przekładka.

* label - tworzy obiekt etykiety.

* listbox - tworzy pole listy tekstowej. Przedmioty są dodawane po definicji widgetu.

* menu - pojedynczy wieloaspektowy widget, który może zawierać różnorodne elementy w różnych stylach menu.

* menubutton - zapewnia klikalny interfejs dla rozwijania menu rozwijanego.

* message - tworzy widżet okna wyświetlającego tekst z funkcjami takimi jak justowanie i zawijanie słów.

* radiobutton - tworzy przycisk radiowy, który może być zbiorem powiązanym z określoną zmienną.

* scale - tworzy suwak do wybierania wartości w określonym zakresie i rozdzielczości.

* scrollbar - generuje widżet (suwak) do zmiany fragmentu materiału (zazwyczaj tekstu lub rysunku) w powiązanym widżecie.

* text - tworzy widget wyświetlający jeden lub więcej wierszy tekstu i umożliwia edycję tekstu

* toplevel - tworzy nowe okno ekranu (na pulpicie X).

**Demonstracja TK**

Jako przykład prostego kodu TK, poniższa lista generuje obraz po lewej stronie. Kod procedury wywołanej przyciskiem OK i przykładowe dane wyjściowe są wyświetlane w oknie tekstowym obrazu.

~/tcltk$ wish

% . configure -width 200 -height 400

% label .header -text "Tk Tutorial Example"

.header

% place .header -x 5 -y 2

% scale .slider -from 1 -to 100 -orient horiz

.slider

% .slider configure -variable SlidVal

% place .slider -x 5 -y 20

% entry .slidbox -width 5 -textvariable SlidVal

.slidbox

% place .slidbox -x 120 -y 38

% radiobutton .one -text "Don't Worry" -variable Mood -value .one

% radiobutton .two -text "Be Happy" -variable Mood -value 2

.two

% place .one -x 5 -y 70

% place .two -x 5 -y 90

% text .twindow -width 22 -height 14 -font {clean -14}

.twindow

% place .twindow -x 5 -y 120

% button .ok -command {process_data $SlidVal} -text "OK"

.ok

% button .cancel -command {exit} -text "Cancel" -background .cancel

% place .ok -x 15 -y 350

% place .cancel -x 120 -y 350

**Polecenia TK, część 1**

Istnieje ponad 20 poleceń TK, które działają, ulepszają lub uzupełniają zestaw widżetów Tk. Należą do nich dzwonek, który dzwoni, w zależności od konfiguracji systemu X Window, który działa. bind tworzy powiązanie między skryptem Tcl a zdarzeniami X; na przykład określone działanie kombinacji klawiszy i myszy. schowek to kolejne z wielofunkcyjnych poleceń Tk - zawiera cały kod do czyszczenia, ładowania i wklejania treści do i ze schowka Tk (który jest różny od wszystkich funkcji schowka dostępnych dla X lub menedżera okien, którym jesteś za pomocą). destroy służy do usunięcia okna i wszystkich jego elementów podrzędnych. Używany na "." (root), usuwa całą aplikację. Zdarzenie to jest potężnym narzędziem do generowania zdarzeń wirtualnego okna i wstawiania ich do kolejki przetwarzania, tak jakby rzeczywiste zdarzenie (np. kliknięcie przycisku) miało miejsce w rzeczywistości. Polecenie font służy do tworzenia określonych instancji czcionek systemowych. Pozwala na lokalne (do skryptu) nazywanie czcionek systemowych, atrybut Presented by developerWorks, źródło świetnych samouczków ibm.com/developerWorks Tcl / Tk quick start Page 21 of 33 modyfikowanie nazwanych

czcionek i "usuwanie" czcionek. Wpisz rodziny czcionek w monicie życzeń, aby wyświetlić listę dostępnych czcionek do użycia. Focus to ważna koncepcja na arenie zarządzania oknami - na każdym ekranie tylko jedno okno może mieć "uwagę" klawiatury i myszy. Polecenie Tk focus służy do kontroli skryptu nad foksem wyświetlacza, wysyłając go do określonych okien. Komplementarna funkcja, grab, pozwala TK zmonopolizować fokus wyświetlania do punktu, w którym zdarzenia poza oknem są zgłaszane w środowisku okna. Jest to przydatne, gdy chcesz wymusić ukończenie opcji przed rozpoczęciem dowolnej innej czynności systemowej.

**Polecenia TK, część 2**

Kontynuując nasz przegląd poleceń Tk, następna jest siatka, interfejs do wzornika geometrii okna Tk. Służy do porządkowania widżetów w oknie w formacie wierszy i kolumn. niżej (i uzupełniającemu podnoszeniu polecenia) pośredniczą w widoczności podokna. Obniżone okno nie zasłania żadnego z nakładających się na siebie okien siostrzanych; podniesione okno jest na górze. Jest to często używane w sytuacjach wyświetlania wielu dokumentów. Wiele widżetów i poleceń TK działa ze wspólnego zestawu standardowych opcji. Można je wyświetlić lub dodać za pomocą polecenia opcji. Do umieszczania widżetów i podokien w oknach są dwa polecenia, które zostały już zademonstrowane: paczka i miejsce. W najprostszym użyciu, pakiet dodaje jeden lub więcej widżetów do okna, i jeśli nie zalecono inaczej, zmniejsza okno wokół tych obiektów, jak widzieliśmy w przykładzie Tk Hello na początku tej sekcji. umieszczaj zestawy i wyświetla obiekty w oknie nadrzędnym, używając względnych lub bezwzględnych miar, na przykład 5 pikseli od lewej strony lub w połowie (0,5) w dół okna. Inne polecenia obejmują wybór, interfejs zestawu narzędzi do wyboru obiektu X; tk, który zapewnia dostęp do wybranych części wewnętrznego stanu interpretera Tk; komenda winfo do pobierania danych o oknach zarządzanych przez Tk; i wm, interfejs do uruchomionego menedżera okien, do ustawiania wielu funkcji z tekstu paska tytułu do wszystkich rodzajów specyfikacji geometrii i ograniczeń

**Prawdziwa (mała) aplikacja TK**

Chcę mieć interfejs do skryptów zmiany sieci LAN, które uruchamiam codziennie. Użyjmy więc Tcl / Tk, aby zbudować małe narzędzie ułatwiające obsługę. Chcę, aby oferował wybory w oparciu o plik konfiguracyjny ASCII, który znajduje się w moim katalogu domowym. Ten plik zawiera dane przedstawione w wykazie, co następuje:

```
# ~/.netsetrc

# 03.26.2001 bilbrey

# space between name and command

Home /usr/local/bin/nethome

Office /usr/local/bin/netoffice

Admin /usr/local/bin/netadmin
```

Aplikacja odczytuje plik konfiguracyjny i analizuje każdą niepustą, nie komentującą linię dla nazwy przycisku i związanego z nim działania. Chociaż łatwiej byłoby napisać skrypt, definiując trzy przyciski do uruchamiania programów jawnych, to bardziej ogólne rozwiązanie pozwala mi dodać dowolną żądaną funkcję, dodając tylko jedną linię do ~ / .netsetrc. Wadą tego kodu jest to, że nie toleruje źle sformatowanego pliku konfiguracyjnego. Oczekuje on nazwy pojedynczego przycisku, po którym następuje pojedyncza spacja, a następnie polecenie (z ewentualnymi argumentami) do wykonania po naciśnięciu przycisku. Jednak plik konfiguracyjny teoretycznie jest łatwiejszy do utrzymania w linii niż niestrukturalne dane wejściowe użytkownika.

**Przykładowa aplikacja TK**

```
#!/usr/bin/wish
#
# netset.tcl
# 03.26.2001 bilbrey
set ConFile "~/.netsetrc"
if [catch {open $ConFile r} Conf] {
puts stderr "Open $ConFile failed"
return 1
}
# parse config, define buttons
set Bcount 0
while {[gets $Conf Cline] >= 0} {
if {1 == [string match #* $Cline]} continue
if {[string length $Cline] < 4} continue
set Nend [string wordend $Cline 0]
incr Nend -1
set Bname [string range $Cline 0 $Nend]
set Cbeg [expr $Nend + 2]
set Bcomd "exec "
append Bcomd [string range $Cline $Cbeg end]
incr Bcount
set NextBut "button$Bcount"
button .$NextBut -text $Bname -command $Bcomd
}
if {$Bcount == 1} {
puts stderr "No buttons defined"
return 2
}
# display buttons
while {$Bcount >= 1} {
```

```
set NextBut "button$Bcount"

pack .$NextBut -padx 10 -pady 10

incr Bcount -1

}

button .exit -text Exit -command {exit}

pack .exit -padx 10 -pady 10
```

**Co to jest Expect?**

Expect jest rozszerzeniem języków Tcl i Tk. Expect zapewnia prosty, ale potężny interfejs do automatyzacji skryptowania interaktywnych programów. Dodatkowo Expect ułatwia osadzanie interaktywnych aplikacji w GUI. Oczekiwany rozwój jest zbieżny z pojawieniem się Tcl / Tk i zależy od obu w obecnej wersji, 5.32. Autorem Expect jest Don Libes, który pracuje w amerykańskim Narodowym Instytucie Standardów i Technologii (NIST). Strona główna dla Expect jest hostowana na serwerach NIST. (Oczekiwania i wszelkie powiązane komercyjne lub niekomercyjne produkty nie są wyraźnie promowane przez NIST.) W poniższych panelach przyjrzymy się kilku przykładom skryptów Expect pobranych z katalogu przykładowego kodu źródłowego, wraz z krótkim opisem przegląd jego składni poleceń. Dlaczego warto się czegoś nauczyć o Expect? Cytując z pracy Dona, "Używanie zadań do automatyzacji zadań administracyjnych systemu" (USENIX LISA Conference, październik 1990) "... wynikiem jest to, że zestaw narzędzi administratora systemu UNIX jest wypełniony przedstawicielami niektórych z najgorszych interfejsów użytkownika, jakie kiedykolwiek widziano Podczas gdy tylko całkowite przeprojektowanie pomoże rozwiązać wszystkie te problemy, można się spodziewać, że spotka się z bardzo wieloma problemami. "

**Pobieranie RFC z Expect**

Jako wprowadzenie do Expect, sprawdź poniższy przykład. Jest tylko nieznacznie zmodyfikowana w stosunku do wersji znalezionej w katalogu przykładowym ze standardowej dystrybucji źródłowej Expect, podobnie jak wszystkie przykłady w tej sekcji. Przejdźmy przez kod ...

```
#!/usr/local/bin/expect --

< ftp-rfc #># ftp-rfc -index

# retrieves an rfc (or the index) from uunet

exp_version -exit 5.0

if {$argc!=1} {

send_user "usage: ftp-rfc \[#] \[-index]\n"

exit

}

set file "rfc$argv.Z"

set timeout 60

spawn ftp ftp.uu.net
```

```
expect "Name*:"

send "anonymous\r"

expect "Password:"

send "bilbrey@orbdesigns.com\r"

expect "ftp>"

send "binary\r"

expect "ftp>"

send "cd inet/rfc\r"

expect "550*ftp>" exit "250*ftp>"

send "get $file\r"

expect "550*ftp>" exit "200*226*ftp>"

close

wait

send_user "\nuncompressing file - wait...\n"

exec uncompress $file
```

Ten program automatyzuje pobieranie FTP dokumentów IETF RFC (Request For Comment) z archiwum UUNet. Pierwsza linia skryptu wywołuje powłokę Expect. Zauważ, że podałem pełną ścieżkę do pliku wykonywalnego. To najbezpieczniejsze, ponieważ trudno jest poznać środowisko ścieżki danego użytkownika. Skrypt najpierw sprawdza wersję Expect, a następnie drukuje komunikat o użyciu, chyba że podano poprawną liczbę argumentów. Następnie ustawiana jest wartość limitu czasu, aby uniemożliwić skryptowi Expect blokowanie zasobów systemowych, jeśli sesja FTP uruchomiona w następującym wierszu nie połączy się poprawnie. Większość salda skryptu to zestawy par poleceń oczekiwanie / wysyłanie. Każde oczekiwane polecenie czeka na określone wyjście z programu spawnowanego (w tym przypadku ftp), a następnie wysyła poprawną odpowiedź. Zauważ, że istnieją dwie pułapki dla kodów błędów ftp po cd i pobierz instrukcje. W każdym przypadku kod błędu 550 jest dopasowywany do pierwszego, a jeśli jest prawdziwy, to skrypt wychodzi. W przeciwnym razie, postępując zgodnie z kodem 250 (wskazującym na sukces), spróbuj przewidzieć kolejne polecenie. Po otrzymaniu dokumentu skrypt wydaje polecenie zamknięcia sesji ftp. Komenda wait przechowuje przetwarzanie skryptów do momentu zakończenia działania ftp. Wreszcie skrypt wysyła wiadomość do użytkownika, dekompresuje pobrany RFC (lub indeks rfc), a następnie domyślnie wychodzi, nie jawnie.

**Klucze do Expect, część 1**

Istnieją cztery kluczowe polecenia w Expect (język, z wielką literą "E"). Pierwszym jest oczekiwanie (polecenie, małe "e"), które wyszukuje wzorce i wykonuje polecenia, jeśli zostanie dopasowany. Dla każdego polecenia oczekiwać może istnieć kilka grup, z których każda składa się z flag opcji, wzorca do dopasowania i polecenia lub zbioru poleceń do wykonania. oczekiwać, że "nasłuchuje" domyślnie na SDTOUT i STDERR, dopóki nie zostanie wykonany mecz lub upłynął limit czasu.

```
#!../expect -f
```

```
# wrapper to make passwd(1) be non-interactive

# username is passed as 1st arg, passwd as 2nd

set password [lindex $argv 1]

spawn passwd [lindex $argv 0]

expect "password:"

send "$password\r"

expect "password:"

send "$password\r"

expect eof
```

Wzorce są domyślnie dopasowywane przy użyciu mechanizmu dopasowywania ciągów Tcl, który implementuje globowanie, podobne do dopasowywania wzorca C-shell. Flaga -re wywołuje dopasowanie regexp, a -ex wskazuje, że dopasowanie powinno być dokładne, bez symboli wieloznacznych i zmiennych. Inne opcjonalne flagi, których można się spodziewać, obejmują -i, aby wskazać, który proces ma monitorować i -nocase, co wymusza przetwarzanie danych wyjściowych na małe litery przed dopasowaniem. Aby uzyskać pełne specyfikacje, wpisz man oczekiwać w wierszu polecenia, aby wyświetlić dokumentację strony podręcznika systemowego dla Expect. Drugim ważnym poleceniem jest send, który służy do generowania danych wejściowych dla procesu monitorowanego przez skrypt Expect. send zawiera opcje wysyłania do określonego sprowadzonego procesu (-i), wysyłanie powoli (-s, aby nie przekroczyć bufora, na przykład w komunikacji szeregowej) i kilka

**Klucze do spodziewania się, część 2**

Poniżej znajduje się skrypt o nazwie carpal, kolejny przykład ze źródła Expect

```
#!/usr/local/bin/expect

# Script to enforce a 10 minute break

# every half hour from typing -

# Written for someone (Uwe Hollerbach)

# with Carpal Tunnel Syndrome.

# If you type for more than 20 minutes

# straight, the script rings the bell

# after every character until you take

# a 10 minute break.

# Author: Don Libes, NIST

# Date: Feb 26, '95

spawn $env(SHELL)

# set start and stop times
```

```
set start [clock seconds]

set stop [clock seconds]

# typing and break, in seconds

set typing 1200

set notyping 600

interact -nobuffer -re . {

set now [clock seconds]

if {$now-$stop > $notyping} {

set start [clock seconds]

} elseif {$now-$start > $typing} {

send_user "\007"

}

set stop [clock seconds]

}
```

spawn to polecenie Expect, które służy do tworzenia nowego procesu. Pojawił się w każdym przykładzie, z którego korzystaliśmy. W powyższym przykładzie ciągnie ścieżkę do domyślnego pliku wykonywalnego powłoki i tworzy nową instancję. W ten sposób spawn zwraca identyfikator procesu, ustawiony w zmiennej spawn_id. Można go zapisać i ustawić w skrypcie, co daje możliwość kontroli procesu. interact to polecenie, którego użycie Expect używa do otwarcia komunikacji między użytkownikiem a procesem zrodzonym. Flaga -nobuffer wysyła znaki pasujące do wzorca bezpośrednio do użytkownika. -re mówi interakcji, aby użyć następującego wzoru jako standardowego wyrażenia regularnego, a "." Jest to wzorzec, dopasowujący każdy znak, jaki się pojawia. W trybie interaktywnym, przekierowanie oczekiwania strumieni STDOUT i STDERR jest również domyślnie zwracane użytkownikowi.

**Co możesz zrobić dzięki Expect**

Kiedy twój skrypt wywołuje interaktywny program, domyślnie Expect przechwytuje wszystkie wejścia i wyjścia (STDIN, STDOUT i STDERR). Pozwala to na szukanie wzorców, które pasują do wyjścia programu i wysyłanie danych wejściowych do zainicjowanego procesu w celu symulacji interakcji użytkownika. Dodatkowo, Expect może przekazać kontrolę nad procesem dla użytkownika, jeśli tak poinstruuje, lub przejąć kontrolę na żądanie. Nie tylko te cechy sprawiają, że Expect jest niezwykle przydatny do wykonywania typowych zadań administracyjnych, ale okazuje się, że Expect jest całkiem dobry do tworzenia skryptów testowych do przeprowadzania walidacji I / O podczas tworzenia programu. W końcu jest niesamowicie przydatny program, autoexpect. Sama skrypt Expect, autoexpect monitoruje interaktywny program w linii poleceń, generując skrypt Expect, który dokładnie powiela tę interakcję. Teraz, gdy zwykle nie jest to tylko to, co jest potrzebne, łatwo jest uzyskać wyniki kilku sesji automatycznego wykrywania, uogólnić wzorce oczekiwania, a następnie wyciąć i wkleić je do pożądanej konfiguracji. Zostało napisane w więcej niż jednym miejscu, że najlepszym narzędziem nauki dla Expect jest automatyczne uruchamianie i odtwarzanie wyników.

**Rozszerzenia Tcl / Tk**

**Wprowadzenie do Tcl / Tk**

Oczekiwano jedynie pierwszego pojawienia się powodzi rozszerzeń Tcl / Tk. Kilka z nich ma ogólną użyteczność, wiele innych jest specyficznych dla oprogramowania lub aplikacji. Ponieważ program Tcl Developer Xchange pozostaje centralnym repozytorium dla wszystkich rzeczy Tcl / Tk, strona rozszerzeń na tej stronie jest cennym źródłem do dalszych poszukiwań. W następnych panelach rzucimy okiem na kilka głównych rozszerzeń, dotykając ich istotnych cech i atrakcji.

**[incr Tcl], [incr Tk] i więcej ...**

Wprowadzony w 1993 r. Zapewnia obiektową funkcję Tcl / Tk. [incr Tcl] udostępnia funkcje obiektu, klasy i przestrzeni nazw. Te cechy ułatwiają tworzenie dużych projektów z Tcl z enkapsulacją danych, składem i dziedziczeniem. Odbywa się to za pomocą: classname, objname i delete są komendami obiektu. Tworzenie i edycja klas odbywa się za pomocą poleceń body, class i configbody. Inne różne polecenia to kod, zespół, wyszukiwanie, lokalne i zakres. Jego graficznym odpowiednikiem jest. To narzędzie rozszerza się do GUI o tę samą funkcjonalność OO, która jest potrzebna do zapewnienia łatwości skalowania i ukrywania danych, co znacznie ułatwia partycjonowanie dużych zadań programowania. [incr Tk] udostępnia nowe klasy bazowe: itk :: Archetype, itk :: Widget i itk :: Toplevel. Te klasy są uzupełnione kompletnym zestawem metod. Zbudowany na fundamencie [incr Tk], teraz jest tak zwany zestaw Mega-Widget. To narzędzie pozwala definiować i wyświetlać złożone obiekty, takie jak pola wyboru plików, z ogromną łatwością. Pożyczone na stronach internetowych [incr Widgets] obraz po lewej jest tworzony za pomocą następujących poleceń poleceń: fileselectiondialog .fsd; .fsd aktywować.

**BLT i Tix**

W interesie bycia równie niesprawiedliwym (nie czyniąc żadnej z nich sprawiedliwością) parą niezwykłych rozszerzeń graficznych i graficznych Tcl / Tk, pozwól, że przedstawię cię najpierw BLT (ftp://tcl.activestate.com/pub/tcl/ blt /). BLT to rozszerzenie TK, które zapewnia funkcje do łatwego tworzenia widgetów wielu elementów, które są bardzo trudne w zwykłym TK. Polecenia BLT obejmują tabelę, wykres, wykresy słupkowe, wektor, splajn, zajęte, bgexec, drag & drop, htext, bitmap, winop, watch i bltdebug. Następna jest Tix, która oznacza eXtension interfejsu TK. Tix jest obecnie w wersji 4.0 i zawiera zestaw 43 poleceń, z których większość to Mega-widżety lub komponenty do budowania megapikseli wraz z kilkoma narzędziami. Strona internetowa Tix, pod adresem http://tix.sourceforge.net/, twierdzi: "Dzięki Tix możesz zapomnieć o niepoważnych szczegółach widżetów TK i skoncentrować się na rozwiązywaniu swoich problemów". Łatwo jest zobaczyć podstawę tego twierdzenia, kiedy możesz szybko stworzyć użyteczne interfejsy z poleceniami takimi jak tixDirList, tixFileSelectDialog, tixPopupMenu, tixScrolledWindow i wieloma innymi.

**TclX**

Extended Tcl, TclX, jest czymś więcej niż kolejnym "rozszerzeniem". Według słów autorów "Extended Tcl jest zorientowany na programowanie systemów i duży rozwój aplikacji, TclX zapewnia dodatkowe interfejsy do natywnego systemu operacyjnego, jak również wiele nowych konstrukcji programistycznych, narzędzi do manipulowania tekstem i możliwości debugowania." Internetowy dom TclX znajduje się pod adresem http://www.neosoft.com/TclX/. Wiele z oryginalnych funkcji TclX znalazło się w głównej dystrybucji Tcl w ciągu ostatnich kilku lat. Jednakże zespół TclX pozostał na piłce, dodając takie funkcje jak dynamiczne ładowanie bibliotek i pakietów, obsługę programowania sieciowego, procedury zapewniające dostęp do poleceń do funkcji matematycznych normalnie wywoływanych przez expr i wiele więcej. TclX jest dołączony jako pakiet, który może być opcjonalnie instalowany z najbardziej standardowym Linuxem Prezentowany przez developerWorks, źródło

świetnych samouczków ibm.com/developerWorks Tcl / Tk quick start Page 30 of 33 dystrybucji. Alternatywnie można go skompilować ze źródła w połączeniu z Tcl i Tk. Jedną z bardzo miłych cech ostatnich wersji TclX jest program o nazwie tclhelp, który jest przeglądarką pomocy Tcl i Tk, która jest bardzo przydatna do przeglądania. Wysoce polecany.