Basic Computing Tools

Group 22

1 Bash

1.1 Introduction

Bash is a Unix shell and command language written by Brian Fox for the GNU Project as a free software replacement for the Bourne shell. Released in 1989, it has been distributed widely as the shell for the GNU operating system and as a default shell on Linux and OS X. It was announced during the 2016 Build Conference that Windows 10 has added a Linux subsystem which fully supports Bash and other Ubuntu binaries running natively in Windows. In the past, and currently, it has also ported to Microsoft Windows and distributed with Cygwin and MinGW, to DOS by the DJGPP project, to Novell NetWare and to Android via various terminal emulation applications. In the late 1990s, Bash was a minor player among multiple commonly used shells; at present Bash has overwhelming favor.

1. For quick display of files:

```
\$ cat helloworld.sh
#!/bin/bash
echo Hello World
```

1.2 GREP

grep is a command-line utility for searching plain-text data sets for lines matching a **regular expression**. Grep was originally developed for the Unix operating system, but is available today for all Unix-like systems. Its name comes from the ed command g/re/p (**g**lobally search a **re**gular expression and **p**rint), which has the same effect: doing a global search with the regular expression and printing all matching lines.

Some basic grep commands are as follows:

1. For basic string search:

```
\$ grep "literal\_string" filename
```

2. For case insensitive search:

```
\$ grep -i "string" filename
```

3. For regular expressions:

```
\$ grep "REGEX" filename
```

4. To display N lines after match:

```
\$ grep -A <N> "string" filename
```

5. To display N lines before match:

```
\$ grep −B <N> "string" filename
```

6. To display lines which do not contain match:

```
\$ grep −v −e "pattern" −e "pattern" filename
```

7. Counting number of matches:

```
\$ grep −c "pattern" filename
```

8. To display N lines before match:

```
\$ grep −B <N> "string" filename
```

1.3 SED

sed (stream editor) is a Unix utility that parses and transforms text, using a simple, compact programming language. **sed** was developed from 1973 to 1974 by Lee E. McMahon of Bell Labs, and is available today for most operating systems. sed was based on the scripting features of the interactive editor ed ("editor", 1971) and the earlier qed ("quick editor", 196566). sed was one of the earliest tools to support regular expressions, and remains in use for text processing, most notably with the substitution command. Other options for doing "stream editing" include AWK and Perl.

1. To match files and replace:

```
sed −e s/<find expression>/<replace expression>/ filename
```

2. To use the match as a part of replace string, we can use the following command:

```
sed −n −e 's/United States/& of America/p' country.txt
United States of America
```

3. To convert lower case letters to upper case:

```
sed 'y/ul/UL/' file.txt
```

1.4 AWK

AWK is an interpreted programming language designed for text processing and typically used as a data extraction and reporting tool. It is a standard feature of most Unix-like operating systems.

The AWK language is a data-driven scripting language consisting of a set of actions to be taken against streams of textual data either run directly on files or used as part of a pipeline for purposes of extracting or transforming text, such as producing formatted reports. The language extensively uses the string datatype, associative arrays (that is, arrays indexed by key strings), and regular expressions. While AWK has a limited intended application domain and was especially designed to support one-liner programs, the language is Turing-complete, and even the early Bell Labs users of AWK often wrote well-structured large AWK programs.

1. Printing columns:

```
awk '/a/ {print $3 "\t" $4}' marks.txt
```

2. Adding variables:

```
awk '/a/{++cnt} END {print "Count=", cnt}' marks.txt
```

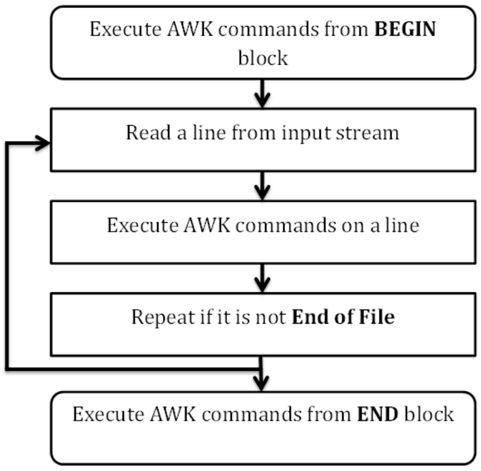
Execute AWK commands from **BEGIN** block
Read a line from input stream
Execute AWK commands on a line
Repeat if it is not **End of File**
Execute AWK commands from **END** block

Figure 1: Awk Workflow

2 Latex

LaTeXis a typesetting system that is very suitable for producing scientific and mathematical documents of high typographical quality. It is also suitable for producing all sorts of other documents, from simple letters to complete books.

LaTeXenables authors to typeset and print their work at the highest typographical quality, using a pre-defined, professional layout. LaTeXwas originally written by Leslie Lamport.

LaTeXcommands are case sensitive, and take one of the following two formats:

1. They start with a backslash \and then have a name consisting of letters only. Command names are terminated by a space, a number or any other 'non-letter.'
2. They consist of a backslash and exactly one non-letter.
3. Many commands exist in a 'starred variant' where a star is appended to the command name.

2.1 Input File Structure

When LaTeXprocesses an input file, it expects it to follow a certain structure. Thus every input file must start with the command.

```
\documentclass{...}
```

This specifies what sort of document you intend to write. After that, add commands to influence the style of the whole document, or load packages that add new features to the LaTeXsystem. To load such a package you use the command.

```
\usepackage{...}
```

When all the setup work is done, you start the body of the text with the command.

```
\begin{docu\section{Latex}
\LaTeX is a typesetting system that is very suitable for producing scientific and mathematical document
\\
\LaTeX enables authors to typeset and print their work at the highest typographical quality, using a pr

\LaTeX commands are case sensitive, and take one of the following two formats:
\begin{enumerate}
\item They start with a backslash \textbackslash and then have a name consisting of
letters only. Command names are terminated by a space, a number or
any other 'non-letter.'
\item They consist of a backslash and exactly one non-letter.
\item Many commands exist in a 'starred variant' where a star is appended
to the command name.
\end{enumerate}
\subsection{Input File Structure}
When \LaTeX processes an input file, it expects it to follow a certain structure. Thus every input file
\begin{verbatim}
\documentclass{...}
```

This specifies what sort of document you intend to write. After that, add commands to influence the style of the whole document, or load packages that add new features to the LaTeXsystem. To load such a package you use the command.

```
\usepackament}
```

Now you enter the text mixed with some useful LaTeXcommands. At the end of the document you add the

```
\end{document}
```

2.2 Titles, Chapters, and Sections

To help the reader find his or her way through your work, you should divide it into chapters, sections, and subsections. LaTeXsupports this with special commands that take the section title as their argument. It is up to you to use them in the correct order.
The following sectioning commands are available for the **article** class:

```
\section{...}
\subsection{...}
\subsubsection{...}
\paragraph{...}
\subparagraph{...}
```

If you want to split your document into parts without influencing the section or chapter numbering use:

```
\part{...}
```

LaTeXcreates a table of contents by taking the section headings and page numbers from the last compile cycle of the document. The command

```
\tableofcontents
```

The title of the whole document is generated by issuing a command:

```
\maketitle
```

a footnote is printed at the foot of the current page. Footnotes should always be put after the word or sentence they refer to. Footnotes referring to a sentence or part of it should therefore be put after the comma or period.

```
\footnote{footnote text}
```

In scientific publications it is customary to start with an abstract which gives the reader a quick overview of what to expect. LaTeXprovides the **abstract** environment for this purpose. Normally **abstract** is used in documents typeset with the article document clas

```
\begin{abstract}
The abstract abstract.
\end{abstract}
```

2.3 Tables

The tabular environment can be used to typeset beautiful tables with optional horizontal and vertical lines. LaTeXdetermines the width of the columns automatically.

Below you can see the simplest working example of a table :

```
\begin{center}
\begin{tabular}{ c c c }
cell1 & cell2 & cell3 \\
cell4 & cell5 & cell6 \\
cell7 & cell8 & cell9
\end{tabular}
\end{center}
```

```
cell1  cell2  cell3
cell4  cell5  cell6
cell7  cell8  cell9
```

```
The tabular environment is more flexible, you can put separator lines in between each column.\\
```

It was already said that the tabular environment is used to type tables. To be more clear about how it works below is a description of each command.

```
{ |c|c|c| }
```

This declares that three columns, separated by a vertical line, are going to be used in the table. Each c means that the contents of the column will be centred, you can also use r to align the text to the right and l for left alignment.

```
\hline
```

This will insert a horizontal line on top of the table and at the bottom too. There is no restriction on the number of times you can use

```
\hline
```

.

```
cell1 & cell2 & cell3 \\
Each & is a cell separator and the double-backslash \\ sets the end of this row.
```

2.4 Typesetting Mathematical Formulae

A mathematical formula can be typeset in-line within a paragraph (text style), or the paragraph can be broken and the formula typeset separately (display style). Mathematical equations within a paragraph are entered between $ and $:

Let's see an example of the **inline** mode:

```
In physics, the mass-energy equivalence is stated by the equation $E=mc^2$,
discovered in 1905 by Albert Einstein.
```

The displayed mode has two versions: numbered and unnumbered.

```
The mass-energy equivalence is described by the famous equation

$$E=mc^2$$

discovered in 1905 by Albert Einstein.
In natural units ($c$ = 1), the formula expresses the identity

\begin{equation}
E=m
\end{equation}
```

The *amsmath* package provides a handful of options for displaying equations. You can choose the layout that better suits your document, even if the equations are really long, or if you have to include several equations in the same line.

3 Octave

GNU Octave is software featuring a high-level programming language, primarily intended for numerical computations. It provides a command-line interface for solving linear and nonlinear problems numerically, and for performing other numerical experiments using a language that is mostly compatible with MATLAB. It may also be used as a batch-oriented language. It is part of the GNU Project, it is free software under the terms of the GNU General Public License.

Octave is one of the major free alternatives to MATLAB, others being Julia and Scilab. These however put less emphasis on (bidirectional) syntactic compatibility with MATLAB than Octave does.

3.1 MATLAB compatibility

Octave has been built with MATLAB compatibility in mind, and shares many features with MATLAB:

1. Matrices as fundamental data type.
2. Built-in support for complex numbers.
3. Powerful built-in math functions and extensive function libraries.
4. Extensibility in the form of user-defined functions.

Due to this, it is easy to look for octave's documentation on Mathworks at `http://in.mathworks.com/`

4 gnuplot

Gnuplot is a portable command-line driven graphing utility for Linux, OS/2, MS Windows, OSX, VMS, and many other platforms. The source code is copyrighted but freely distributed (i.e., you don't have to pay for it). It was originally created to allow scientists and students to visualize mathematical functions and data interactively, but has grown to support many non-interactive uses such as web scripting. It is also used as a plotting engine by third-party applications like Octave. Gnuplot has been supported and under active development since 1986.

A simple example which plots the the three graphs mentioned on the same co-ordinate system:

```
set title "Some math functions"
set xrange [-10:10]
set yrange [-2:2]
set zeroaxis
plot (x/4)**2, sin(x), 1/x
```

The supported functions include:

abs(x)	absolute value of x, $|x|$
acos(x)	arc-cosine of x
asin(x)	arc-sine of x
atan(x)	arc-tangent of x
cos(x)	cosine of x, x is in radians.
cosh(x)	hyperbolic cosine of x, x is in radians
erf(x)	error function of x
exp(x)	exponential function of x, base e
inverf(x)	inverse error function of x
invnorm(x)	inverse normal distribution of x
log(x)	log of x, base e
log10(x)	log of x, base 10
norm(x)	normal Gaussian distribution function
rand(x)	pseudo-random number generator
sgn(x)	1 if $x > 0$, -1 if $x < 0$, 0 if $x = 0$
sin(x)	sine of x, x is in radians
sinh(x)	hyperbolic sine of x, x is in radians
sqrt(x)	the square root of x
tan(x)	tangent of x, x is in radians
tanh(x)	hyperbolic tangent of x, x is in radians

4.1 GNUPLOT SCRIPTS

Sometimes, several commands are typed to create a particular plot, and it is easy to make a typographical error when entering a command. To stream- line your plotting operations, several Gnuplot commands may be combined into a single script file. For example, the following file will create a customized display of the force-deflection data:

```
# Gnuplot script file for plotting data in file "force.dat"
# This file is called   force.p
set   autoscale                          # scale axes automatically
unset log                                # remove any log-scaling
unset label                              # remove any previous labels
```

```
set xtic auto                          # set xtics automatically
set ytic auto                          # set ytics automatically
set title "Force Deflection Data for a Beam and a Column"
set xlabel "Deflection (meters)"
set ylabel "Force (kN)"
set key 0.01,100
set label "Yield Point" at 0.003,260
set arrow from 0.0028,250 to 0.003,280
set xr [0.0:0.022]
set yr [0:325]
plot    "force.dat" using 1:2 title 'Column' with linespoints , \
"force.dat" using 1:3 title 'Beam' with points
```

4.2 CUSTOMIZING YOUR PLOT

Many items may be customized on the plot, such as the ranges of the axes, the labels of the x and y axes, the style of data point, the style of the lines connecting the data points, and the title of the entire plot.

4.2.1 plot command customization

Plots may be displayed in one of eight styles: lines, points, linespoints, impulses, dots, steps, fsteps, histeps, errorbars, xerrorbars, yerrorbars, xyerrorbars, boxes, boxerrorbars, boxxyerrorbars, financebars, candlesticks or vector To specify the line/point style use the plot command as follows:

```
gnuplot> plot "force.dat" using 1:2 title 'Column' with lines, \
"force.dat" u 1:3 t 'Beam' w linespoints
```

Note that the words: using , title , and with can be abbreviated as: u, t, and w. Also, each line and point style has an associated number.

4.2.2 set command customization

Customization of the axis ranges, axis labels, and plot title, as well as many other features, are specified using the set command. Specific examples of the set command follow. (The numerical values used in these examples are arbitrary.) To view your changes type: replot at the gnuplot> prompt at any time.

```
Create a title:                  > set title "Force-Deflection Data"
Put a label on the x-axis:       > set xlabel "Deflection (meters)"
Put a label on the y-axis:       > set ylabel "Force (kN)"
Change the x-axis range:         > set xrange [0.001:0.005]
Change the y-axis range:         > set yrange [20:500]
Have Gnuplot determine ranges:   > set autoscale
Move the key:                    > set key 0.01,100
Delete the key:                  > unset key
Put a label on the plot:         > set label "yield point" at 0.003, 260
Remove all labels:               > unset label
Plot using log-axes:             > set logscale
Plot using log-axes on y-axis:   > unset logscale; set logscale y
Change the tic-marks:            > set xtics (0.002,0.004,0.006,0.008)
Return to the default tics:      > unset xtics; set xtics auto
```

5 XFig

Xfig is a menu-driven tool that allows the user to draw and manipulate objects interactively under the X Window System. *It runs under X version* and requires a two- or three-button mouse. file specifies the name of a file to be edited. The objects in the file will be read at the start of xfig.
The figure generated by xfig needs to be post processed by an external tool to convert to a different, more usable format like JPEG or PNG. This is usually done with *fig2dev*, a program found in the Transfig package.
XFig was one of the first widely used vector graphics editor (in contrast, programs like photoshop use raster graphics), which means the images are essentially stored in forms of bazie curves, basic shapes and straight lines instead of having separate colors for different pixels. Due to the advent of other programs like Inkscape which used a lot more of today's available hardware, the program XFig now belongs as a part of history and is no longer as popular as it used to be.

6 HTML

7 Git

7.1 Introduction

A **version control system (VCS)** allows you to track the history of a collection of files. It supports creating different versions of this collection. Each version captures a snapshot of the files at a certain point in time and the VCS allows you to switch between these versions. These versions are stored in a specific place, typically called a repository. **Git** is a widely used source code management system for software development. It is a **distributed revision control system** with an emphasis on speed, data integrity, and support for distributed, non-linear workflows. In a distributed version control system each user has a complete local copy of a repository on his individual computer. The user can copy an existing repository. This copying process is typically called cloning and the resulting repository can be referred to as a clone.

7.2 Git terminology

- **Cloning** : The process of copying an existing Git repository is called cloning. After cloning a repository the user has the complete repository with its history on his local machine.
- **Working Tree** : A local repository provides at least one collection of files which originate from a certain version of the repository. This collection of files is called the working tree.
- **Branching** : Git supports branching which means that you can work on different versions of your collection of files. A branch separates these different versions and allows the user to switch between these versions to work on them.
- **Repository** : A repository contains the history, the different versions over time and all different branches and tags. In Git each copy of the repository is a complete repository.
- **Commit** : When you commit your changes into a repository this creates a new commit object in the Git repository. This commit object uniquely identifies a new revision of the content of the repository.

7.3 Setting up a Repository

7.3.1 git init

The git init command creates a new Git repository. It can be used to convert an existing, unversioned project to a Git repository or initialize a new empty repository. Executing git init creates a .git subdirectory in the project root, which contains all of the necessary metadata for the repo.

git init Transform the current directory into a Git repository. This adds a .git folder to the current directory and makes it possible to start recording revisions of the project.

7.3.2 git clone

The git clone command copies an existing Git repository.

git clone ”repo” Clone the repository located at ¡repo¿ onto the local machine. The original repository can be located on the local filesystem or on a remote machine accessible via HTTP or SSH.

7.4 Saving changes

7.4.1 git add

The git add command adds a change in the working directory to the staging area. It tells Git that you want to include updates to a particular file in the next commit. However, git add doesn't really affect the repository in any significant waychanges are not actually recorded until you run git commit.

git add ”file” Stage all changes in ”file” for the next commit.

7.4.2 git commit

The git commit command commits the staged snapshot to the project history. Committed snapshots can be thought of as safe versions of a projectGit will never change them unless you explicity ask it to.

git commit Commit the staged snapshot. This will launch a text editor prompting you for a commit message. After youve entered a message, save the file and close the editor to create the actual commit.

git commit -m ”message” Commit the staged snapshot, but instead of launching a text editor, use ”message” as the commit message.

7.5 Inspecting a repository

7.5.1 git status

The git status command displays the state of the working directory and the staging area. It lets you see which changes have been staged, which havent, and which files arent being tracked by Git. Status output does not show you any information regarding the committed project history.

git status List which files are staged, unstaged, and untracked.

7.6 Viewing old commits

7.6.1 git checkout

The git checkout command serves three distinct functions: checking out files, checking out commits, and checking out branches. In this module, were only concerned with the first two configurations.

Checking out a commit makes the entire working directory match that commit. This can be used to view an old state of your project without altering your current state in any way. Checking out a file lets you see an old version of that particular file, leaving the rest of your working directory untouched.

git checkout master Return to the master branch. Branches are covered in depth in the next module, but for now, you can just think of this as a way to get back to the current state of the project.

git checkout "commit" "file" Check out a previous version of a file. This turns the "file" that resides in the working directory into an exact copy of the one from "commit" and adds it to the staging area.

git checkout "commit" Update all files in the working directory to match the specified commit. You can use either a commit hash or a tag as the "commit" argument.

7.7 Undoing Changes

7.7.1 git revert

The git revert command undoes a committed snapshot. But, instead of removing the commit from the project history, it figures out how to undo the changes introduced by the commit and appends a new commit with the resulting content. This prevents Git from losing history.

git revert "commit" Generate a new commit that undoes all of the changes introduced in "commit", then apply it to the current branch.

7.7.2 git reset

Git reset can be used to remove committed snapshots, although its more often used to undo changes in the staging area and the working directory. In either case, it should only be used to undo local changesyou should never reset snapshots that have been shared with other developers.

git reset "file" Remove the specified file from the staging area, but leave the working directory unchanged. This unstages a file without overwriting any changes.

git reset Reset the staging area to match the most recent commit, but leave the working directory unchanged.

7.7.3 git clean

The git clean command removes untracked files from your working directory. This is really more of a convenience command, since its trivial to see which files are untracked with git status and remove them manually. Like an ordinary rm command, git clean is not undoable, so make sure you really want to delete the untracked files before you run it.

git clean -n Perform a dry run of git clean. This will show you which files are going to be removed without actually doing it.

git clean -f Remove untracked files from the current directory. The -f (force) flag is required unless the clean.requireForce configuration option is set to false (it's true by default).

8 BitBucket